Analysis of optimization and numerical approaches to solve the linear least square problem

Emanuele Cosenza*, Riccardo Massidda†

Department of Computer Science
University of Pisa

* e.cosenza3@studenti.unipi.it, † r.massidda@studenti.unipi.it

***Abstract*—The linear least square problem can be tackled using a wide range of optimization or numerical methods. The L-BFGS method of the class of limited-memory quasi-Newton algorithms has been chosen for the former, whilst the thin QR factorization with Householder reflectors for the latter. Both these algorithms have been implemented from scratch using Python language, to finally experiment over their performances in terms of precision, stability and speed. The accordance of the implementations with the underlying theoretical models is also studied and discussed.**

Introduction

Given a dataset composed by a matrix $\hat{X} \in \mathbb{R}^{m \times n}$ with $m \geq n$ and a vector $y \in \mathbb{R}^m$, the solution of the linear least square (LLS) problem is the vector $w \in \mathbb{R}^n$ that fits best the data assuming a linear function between $\hat{X}$ and y. (Nocedal and Wright 2006, 50) This can be formalized as the following minimization problem:

$$w_* = \min_w \|\hat{X}w - y\|_2^2$$

The matrix $\hat{X}$ is actually composed in the following way:

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

Where $X \in \mathbb{R}^{n \times k}$ is a tall thin matrix, thus $m = k + n$. The LLS problem can be dealt both with iterative methods or with direct numerical methods. One algorithm has been chosen for each of these fields to finally discuss their experimental results.

L-BFGS

The Limited-memory BFGS, L-BFGS, is an iterative method of the quasi-Newton limited-memory class. This method is a variation of the BFGS method, with which it shares the update rule. At the $i+1$-th iteration the point is updated as follows:

$$w_{i+1} = w_i - \alpha_i H_i \nabla f_i$$

The smaller memory requirements of this variation are due to the fact that the Hessian approximation H_i is stored implicitly, and built over a fixed number of vector pairs $\{s_j, y_j\}$ of the previous t iterations and an initial matrix H_i^0. Where

$$s_i = w_{i+1} - w_i, \quad y_i = \nabla f_{i+1} - \nabla f_i$$

$$V_i = I - \rho_i y_i s_i^T, \quad \rho_i = \frac{1}{y_k^T s_k}$$

so that H_i satisfies the following condition

$$\begin{aligned} H_i = {} & (V_{i-1}^T \dots V_{i-t}^T) H_i^0 (V_{i-t} \dots V_{i-1}) \\ & + \rho_{i-t} (V_{i-1}^T \dots V_{i-t}^T + 1) s_{i-t} s_{i-m}^T (V_{i-t+1} \dots V_{i-1}) \\ & + \rho_{i-t+1} (V_{i-1}^T \dots V_{i-t}^T + 2) s_{i-t+1} s_{i-t+1}^T (V_{i-t+2} \dots V_{i-1}) \\ & + \dots \\ & + \rho_{i-1} s_{i-1} s_{i-1}^T \end{aligned}$$

Different strategies to initialize the H_i^0 matrix are proposed in the literature, and so they will be tested experimentally. Finally, the step size α_i is found by performing an inexact line search based on the Armijo-Wolfe conditions.

Thin QR factorization

For the numerical counterpart, the thin QR factorization with Householder reflectors has been implemented as described in (Trefethen and Bau 1997).

By using the Householder QR factorization, the matrix $R \in \mathbb{R}^{m \times n}$ is constructed in place of $\hat{X}$ and the n reflection vectors $v_1, \dots, v_n$ are stored. The reduced matrix $\hat{R} \in \mathbb{R}^{n \times n}$ is trivially obtainable by slicing as in $\hat{R} = R_{1:n,1:n}$, noting that constructing directly the reduced matrix would yield no significant advantage since we are already using $O(mn)$ space to store the reflection vectors.

By using the Householder vectors it is also possible to implicitly compute $\hat{Q}^T y$ to finally obtain w_* by back substitution over the upper-triangular system $\hat{R}w = \hat{Q}^T y$.

Algorithmic analysis

Convergence of L-BFGS

Liu and Nocedal (1989) define three necessary assumptions to prove that the L-BFGS algorithm globally converges and that there exists a constant $0 \leq r < 1$ such that

$$f(w_i) - f(w_*) \leq r^i (f(w_0) - f(w_*))$$

so that the sequence $\{w_i\}$ converges R-linearly.

Firstly the objective function f should be twice continuously differentiable. Given the formulation of the least squares problem this is immediately true, the gradient and the Hessian of the objective function are definable as in:

$$\nabla f(w) = \hat{X}^T (\hat{X} w - y)$$

$$\nabla^2 f(w) = \hat{X}^T \hat{X}$$

Moreover the Hessian can be proven to be positive definite, as can be easily seen by rearranging it in the following way:

$$\begin{aligned} \nabla^2 f(w) &= \hat{X}^T \hat{X} \\ &= |XI| \begin{vmatrix} X^T \\ I \end{vmatrix} \\ &= XX^T + I \end{aligned}$$

The matrix XX^T is positive semi-definite, since $\forall z : z^T XX^T z = \|X^T z\| \geq 0$, therefore all the eigenvalues of the matrix are non-negative. Furthermore, according to the spectral theorem, since XX^T is symmetric, there exists U orthogonal matrix and D diagonal containing the eigenvalues of XX^T.

$$\begin{aligned} \nabla^2 f(x) &= XX^T + I \\ &= UDU^T + I \\ &= UDU^T + UIU^T \\ &= U(D + I)U^T \end{aligned}$$

The eigenvalues of the Hessian are contained in $D + I$ and all of them are positive, therefore $\nabla^2 f(w)$ is positive definite.

Being the Hessian positive definite, the objective function f is a convex function. This comes in handy for the second assumption requiring the sublevel set $D = \{w \in \mathbb{R}^n | f(w) \leq f(w_0)\}$ to be convex. It can be easily proved that if a function is convex all of its sublevel sets are convex sets.

$$\begin{aligned} &\forall x, y \in D, \lambda \in [0, 1] \\ &\qquad \text{f convex} \\ &\qquad \implies f(\lambda x + (1 - \lambda) y) \\ &\qquad \leq \lambda f(x) + (1 - \lambda) f(y) \\ &\qquad \leq \lambda f(w_0) + (1 - \lambda) f(w_0) \\ &\qquad = f(w_0) \\ &\qquad \implies \lambda x + (1 - \lambda) y \in D \end{aligned}$$

The third and last assumption requires the existence of two positive constants M_1 and M_2 such that $\forall z \in \mathbb{R}^n, w \in D$:

$$M_1 \|z\|^2 \leq z^T \nabla^2 f(w) z \leq M_2 \|z\|^2$$

or equivalently

$$M_1 I \preceq \nabla^2 f(w) \preceq M_2 I$$

Since $\nabla^2 f(w)$ is positive definite the previous condition is true for $M_1 = \lambda_{min}$ and $M_2 = \lambda_{max}$, where $\lambda_{min} > 0$.

In the convergence proof the M_2 constant is used to upper bound the trace of the next Hessian substitute H_{i+1}, implying an upper bound for the largest eigenvalue in the sequence of Hessian substitutes.

$$tr(H_{i+1}) \leq tr(H_i^0) + t M_2 \leq M_3$$

On the other hand the M_1 constant is used, to lower bound the determinant of H_{i+1}, implying a lower bound for the smallest eigenvalue in the sequence of Hessian substitutes.

$$det(H_{i+1}) \geq det(H_i^0) + (\frac{M_1}{M_3})^t \geq M_4$$

These two assertions are used to prove the existence of constant $\delta > 0$ such that

$$\forall i : \cos \theta_i = \frac{s_i^T H_i s_i}{\|s_i\| \|H_i s_i\|} \geq \delta$$

where θ_i is the angle between the chosen direction and $-\nabla f(w_i)$. If the constant M_1 was to be equal to zero, it would not be enough to prove the existence of $\delta > 0$ for each step, possibly having directions orthogonal to steepest one. As already pointed out, given that the Hessian is positive definite, its eigenvalues and consequently M_1 are positive.

Other then the three discussed assumptions, the theorem requires for the sequence of initializers $\{\|H_i^0\|\}$ to be bounded. This obviously depends on the initialization technique used to generate H_i^0, various techniques are suggested in the literature such as $H_k^0 = \gamma_k I$ or $H_k^0 = \gamma_k H_0$ where

$$\gamma_k = \frac{s_{k-1}^T y_{k-1}}{\|y_{k-1}\|^2}$$

Other initialization techniques may possibly be tested and evaluated experimentally.

Armijo-Wolfe line search

The convergence proof requires the algorithm to perform a line search respectful of the Armijo-Wolfe conditions. By defining the function ϕ, used to evaluate the value of f at a certain step-size α, the conditions can be defined as follows.

$$\phi(\alpha) = f(w_i + \alpha d_i)$$

$$\phi(\alpha) \leq \phi(0) + \alpha\rho\phi'(0) \tag{A}$$

$$\phi'(\alpha) \geq \sigma\phi'(0) \tag{W}$$

Given the quadratic nature of the least squares problem, it is possible to compute the exact optimal step-size $\bar{\alpha}$ by solving $\phi'(\alpha) = 0$. To simplify the following discussion the objective function $f(w)$ is described in the form $\frac{1}{2}w^T Q w + q^T w + c$ where

$$Q = \hat{X}\hat{X}^T, q^T = -y^T\hat{X}, c = -\frac{1}{2}\|y\|^2$$

Therefore the function $\phi'(\alpha)$ can be defined as follows

$$\phi'(\alpha) = \frac{\partial f(w + \alpha d)}{\partial \alpha} = \nabla f(w)^T d + \alpha d^T Q d$$

where d is the descent direction computed by the L-BFGS algorithm. The optimal step-size is then computable as

$$\bar{\alpha} = -\frac{\nabla f(w)^T d}{d^T Q d}$$

It can be proven that the step size $\bar{\alpha}$ satisfies both (A) for any $\rho \leq \frac{1}{2}$ and (W) for any positive σ. The core steps of the proofs are hereby reported, it should be remarked that these depend on the fact that d is a descent direction, and so that $f(w)^T d < 0$.

Armijo satisfaction proof:

$$\begin{aligned}
\phi(\bar{\alpha}) &\leq \phi(0) + \rho\bar{\alpha}\phi'(0) \\
f(w + \bar{\alpha}d) - f(w) &\leq \rho\bar{\alpha}\nabla f(w)^T d \\
\bar{\alpha}(w^T Q + q^T)d + \frac{\bar{\alpha}^2}{2} d^T Q d + &\leq \rho\bar{\alpha}\nabla f(w)^T d \\
\nabla f(w)^T d + \frac{\bar{\alpha}}{2} d^T Q d + &\leq \rho\nabla f(w)^T d \\
\nabla f(w)^T d - \frac{1}{2}\frac{\nabla f(w)^T d}{d^T Q d} d^T Q d + &\leq \rho\nabla f(w)^T d \\
\rho &\leq \frac{1}{2}
\end{aligned}$$

Wolfe satisfaction proof:

$$\begin{aligned}
\phi'(\alpha) &\geq \sigma\phi'(0) \\
\nabla f(w)^T d + \bar{\alpha} d^T Q d &\geq \sigma\nabla f(w)^T d \\
\nabla f(w)^T d - \nabla f(w)^T d &\geq \sigma\nabla f(w)^T d \\
\sigma &\geq 0
\end{aligned}$$

Analysis of standard and modified thin QR

Ignoring constants, we know from theory that the standard QR factorization algorithm applied on the matrix $\hat{X}$ yields a time complexity of $O(mn^2)$. Actually, given that we are generally dealing with a very tall and thin matrix X, the resulting $\hat{X}$ is going to be squarish ($m \approx n$). This means that we can consider the complexity to be cubic in n.

From now on, we show a way to bring down the time complexity of the algorithm from $O(mn^2)$ to $O(kn^2)$, with $k = m - n$. The resulting modified QR factorization algorithm will become useful when k is much smaller than m, as in our case.

In the standard algorithm, at each step we focus on a single column of the input matrix, constructing a householder vector to zero out all the entries below the diagonal. Following the geometric reasoning in (Trefethen and Bau 1997), this brings the algorithm to depend on m. While this cannot be avoided in general, in our particular case we can be a little bit smarter.

Since the block matrix $\hat{X}$ contains the identity as its lower block, at each step of the algorithm we can just focus on zeroing out the $k = m - n$ entries below the diagonal up to the 1s of the identity. In the modified algorithm then, to obtain R, the matrix $\hat{X}$ is multiplied on the left side by a sequence of matrices $L_i \in \mathbb{R}^{m \times m}$ of the form ($i = 1, \dots, n$):

$$L_i = \begin{bmatrix} I_{i-1} & 0 & 0 \\ 0 & H_i & 0 \\ 0 & 0 & I_{n-i} \end{bmatrix}$$

where $H_i \in \mathbb{R}^{(k+1)\times(k+1)}$ are all Householder reflectors that zero out the k entries in the i-th column of the matrix which is being multiplied by L_i. The resulting Householder vectors will all have constant dimension $k + 1$ and storing them will require $\theta(kn)$ space instead of $O(mn)$.

To derive the time complexity of this phase we can reason as follows. The right side matrix can be divided in three blocks as in $\left[\begin{smallmatrix} A \\ B \\ C \end{smallmatrix}\right]$. When we multiply this matrix by L_i the only relevant operation is the matrix multiplication $H_i B$, which costs $O(kn)$. Since the total number of multiplications is n, the total complexity is $O(kn^2)$.

Since each L_i is orthogonal and symmetric it is then possible to reconstruct Q and the reduced $\hat{Q}$ in the following way:

$$Q = L_1 L_2 \dots L_n$$

$$\hat{Q} = L_1 L_2 \dots L_n \begin{bmatrix} I_n \\ 0 \end{bmatrix}$$

Applying again the reasoning above, the time complexity of these reconstructions is $O(kn^2)$. It follows that the overall time complexity of the modified QR factorization is $O(kn^2)$.

The least squares problem is then solved through back substitution over the upper-triangular system $\hat{R}w = \hat{Q}^T y$. Since the costs for the $\hat{Q}^T y$ product and the back substitution are dominated by the factorization cost, the overall time complexity for solving the least squares problem through QR factorization is $O(mn^2)$ when using the standard algorithm and $O(kn^2)$ when using the modified one.

Stability and accuracy of the QR algorithm

As stated in (Trefethen and Bau 1997, 140), the algorithm obtained by combining the standard QR algorithm, the $Q^T y$ product and back substitution is backward stable in the context of least squares problems.
We claim that the QR factorization step remains backward stable if we consider the modified version described in this report. Without going into details with an extended proof, this can be explained by saying that at each step of the algorithm we apply a transformation L_i doing a smaller number of operations than those of the standard algorithm. Then, since we know that each step of the standard QR factorization is backward stable, this must be true also in the modified version of the algorithm.

Considering that both versions of the QR algorithm are backward stable, the accuracy of the algorithms will depend mostly on the conditioning of the least squares problem at hand. In fact, following from the definition of backward stability, the algorithms will both find exact solutions to slightly perturbed problems, with perturbations of the order of machine precision. This implies that if the conditioning of the problem is high the real solutions to the perturbed problems are inevitably going to be inaccurate. If w_* is the exact solution to the least squares problem and $\tilde{w}_*$ is the solution found with one of the QR based algorithms outlined above, the accuracy of the algorithms will therefore follow the general upper bounds of relative errors found in (Trefethen and Bau 1997, 131): one relative to perturbations of the matrix $\hat{X}$,

$$\frac{\|\tilde{w}_* - w_*\|}{\|w_*\|} \leq (\kappa(\hat{X}) + \kappa(\hat{X})^2 \tan\theta) \frac{\|\delta\hat{X}\|}{\|\hat{X}\|}$$

and one relative to perturbations of the vector y,

$$\frac{\|\tilde{w}_* - w_*\|}{\|w_*\|} \leq \left(\frac{\kappa(\hat{X})}{\cos\theta}\right) \frac{\|\delta y\|}{\|y\|}$$

where θ is the angle such that $\cos\theta = \frac{\|\hat{X}w_*\|}{\|y\|}$.
From these upper bounds we can expect that the algorithm will be more accurate when the angle θ is near 0 and less accurate when it is near $\frac{\pi}{2}$, reminding that in our context the value of θ will depend on the value of the random vector y.

Input data

Because of the bottom I block, the input matrix $\hat{X}$ has n linearly independent rows and consequently full column rank. Therefore, a unique solution is expected from the linear least squares problem regardless of the possible values y. Nonetheless, as discussed in the previous section, the conditioning of the problem is dependent on y, precisely on the angle θ between the image of $\hat{X}$ and y. (Figure 1)

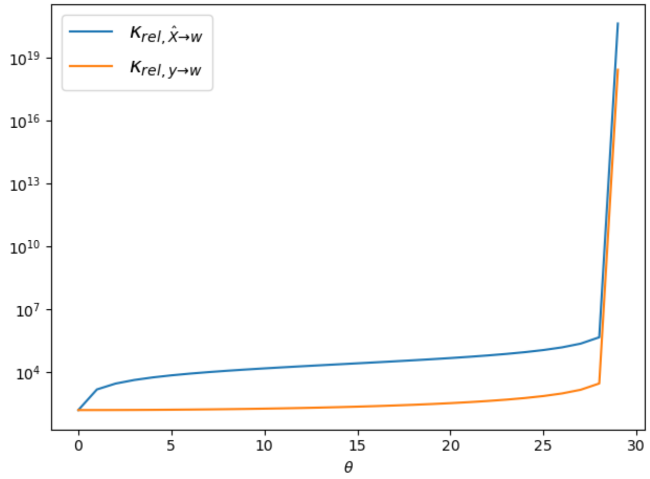


Fig. 1: Relative conditioning of the problem for $\theta \in (0, \frac{\pi}{2})$ (log-scale)

The problem of extracting a random vector y is tackled in two different ways, choosing one or the other depending on the specific experiment. The first way is to extract m random variables using a normal distribution, constructing so the y vector by associating each component to a random variable. This is done by using the Numpy method `random.randn`.

When more control over the conditioning of the problem is instead required, the vector y is extracted considering the θ angle as a constraint. Firstly a vector v perpendicular to the image is required, this can be done by selecting one of the columns of Q_2, from the QR factorization of $\hat{X}$. It is in fact immediate that for each v column of Q_2 and for each $w \in \mathbb{R}^n$

$$v^T \hat{X} w = v^T Q_1 R_1 w = 0$$

since v is orthogonal in respect to each column of Q_1 and different to each of them. This procedure isn't part of the experimental evaluation, therefore the QR factorizer from the Numpy library has been used during this step. By selecting a vector $w \in \mathbb{R}^n$, extracted using multiple normal random variables as previously discussed, it is necessary to constrain the norm of v as in

$$\|v\| = \|\hat{X}w\| \tan\theta$$

The desired vector y it is finally obtainable by summing $\hat{X}w + v$.

Experimental results

The described and implemented techniques have been thoroughly evaluated against themselves and against the NumPy built-in method to solve the linear least squares problem.

...

Many of the described tests report the average of multiple runs, this is aimed to ensure the reproducibility of the experimental results, especially when random behavior plays a role. The number of runs per experiment is fixed, moreover they are executed sequentially to avoid with certainty the effect of parallelization overhead.

For what concerns the optimization techniques, the different tunable parameters are discussed in each experiment when relevant. Default conditions of the L-BFGS algorithm are reported in table I.

TABLE I: Default parameter for the L-BFGS optimizer.

	Parameter	
t	Memory	8
ϵ	Gradient threshold	1e−6
	Max step	2048

For the following experiment, only the behavior of the modified version $QR*$ has been studied. A comparison between the performances of standard QR and $QR*$ is reported in table II.

TABLE II: Average results over multiple runs with normal random extracted y

Model	Time	Residual
QR*	1.62066	21.1557
QR	32.8074	21.1557

Conditioning effects

The angle θ between the image of $\hat{X}$ and the chosen vector y has a great impact on the conditioning of the problem. The average behavior of the different methods against the θ value has been plotted in figure 2. Figure 2a highlights how all the evaluated methods have almost overlapping curves for what concerns the residual of the problem, whilst showing a significant difference for the times in figure 2b. It is evident both in figure 2b and 2c that as the conditioning of the problem worsen the more affected method is the L-BFGS, that isn't able to converge within the limit of the maximum steps allowed.

The same metrics have been studied also away from the extreme regions of θ for what concerns the conditioning. Therefore the average results in the interval $\theta \in (\frac{\pi}{8}, \frac{3\pi}{8})$, also introducing the standard Newton method in the comparison, are reported in table III. As seen in figure 3 in such interval the execution time is constant for each method.

TABLE III: Comparison of different metrics obtained by averaging the results of multiple runs for y such that $\theta \in (\frac{\pi}{8}, \frac{3\pi}{8})$.

Model	Time	Residual	Steps
LLS Numpy	1.7221	45068.4	1
Newton	0.0305718	45068.4	1
LBFGS	0.0818654	45068.4	11.2571
QR*	1.39376	45068.4	1

Initialization L-BFGS

Two different initialization techniques have been tested for the L-BFGS algorithm, precisely the use of $H_k^0 = I$ or $H_k^0 = \gamma_k I$. The BFGS algorithm has been implemented using $H_0 = I$ to offer further insights. Table IV shows how the three variants converge exactly in the same way, whilst it is interesting that the use of I to initialize both L-BFGS and BFGS results in the same sequence of step-size $\{\alpha_i\}$. Regardless of the chosen initialization technique the L-BFGS converges r-linearly as expected. (Figure 4)

Memory in L-BFGS

The L-BFGS algorithm has been tested for different values of memory $t \in (1, n)$, the average results of multiple runs are reported in figure 5. Whilst the memory size had no impact on the residual (figure 5a), a significant reduction of the number of steps occurs even for minimal amounts of memory before stabilizing. Despite the noisy peaks in figure 5b, it is evident how the increase of the memory also as an increasing effect on the time required by the algorithm and consequently on the time per step.

Conclusions

Bibliography

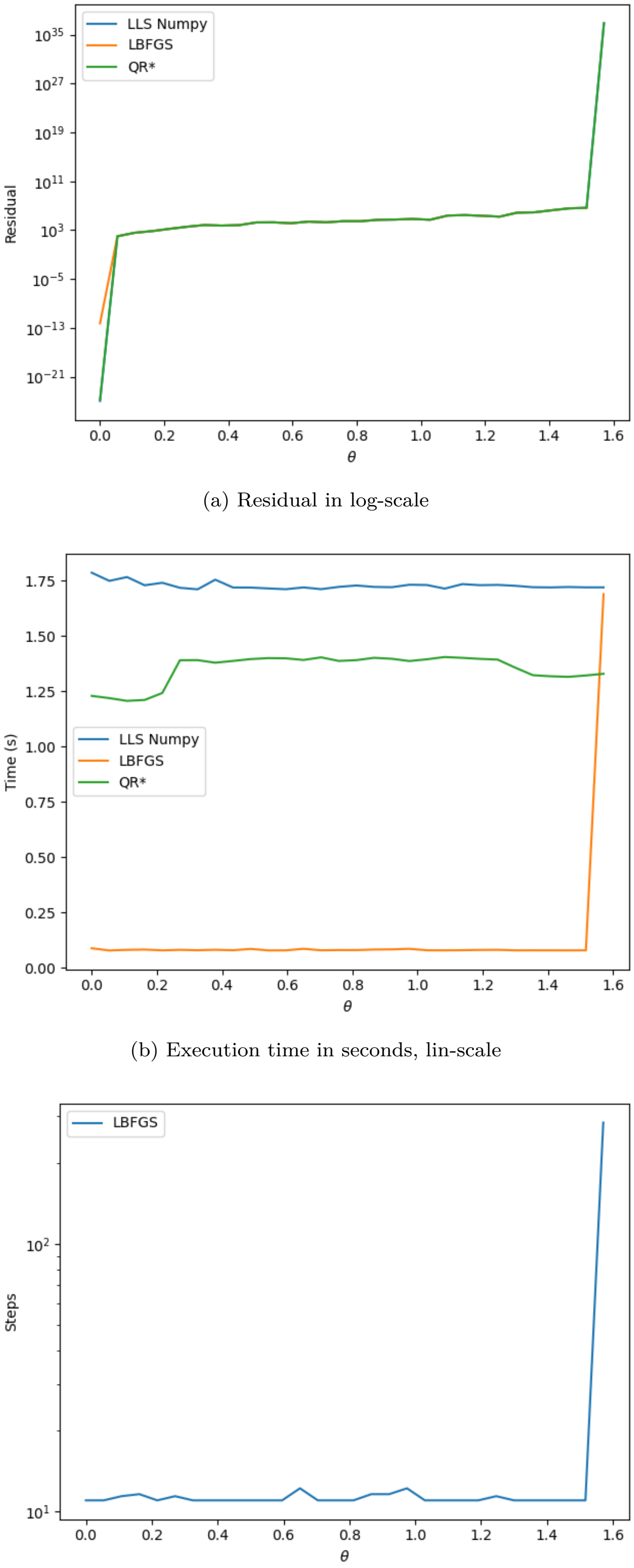
Liu, Dong C., and Jorge Nocedal. 1989. "On the Limited Memory BFGS Method for Large Scale Optimization." *Mathematical Programming* 45 (1-3): 503–28. https://doi.org/10.1007/BF01589116.

Nocedal, Jorge, and Stephen J. Wright. 2006. *Numerical Optimization.* 2nd ed. Springer Series in Operations Research. New York: Springer.

Trefethen, Lloyd N., and David Bau. 1997. *Numerical Linear Algebra.* Philadelphia: Society for Industrial; Applied Mathematics.

TABLE IV: Sample run of L-BFGS initialized with I, γI and BFGS initialized with I.

		γ L-BFGS			I L
Step	α	$\nabla f(w)$	$f(w)$	α	∇
0	4.2441e-05	4.1504e+03	1.0367e+04	4.2441e-05	4.15
1	4.6988e+00	1.7129e+03	7.1077e+03	1.8922e-04	1.71
2	5.8083e+00	6.3167e+02	4.3524e+03	9.3913e-04	6.31
3	2.2707e+00	3.8537e+02	3.6034e+03	1.8772e-03	3.85
4	4.1725e-01	7.4976e+01	3.5187e+03	5.7079e-04	7.49
5	1.1454e+01	1.0631e+02	3.4832e+03	6.2994e-03	1.06
6	1.0217e+01	2.8971e+02	3.2416e+03	2.1379e-02	2.89
7	1.1427e+01	5.0177e+02	8.0810e+02	2.8993e-02	5.01
8	4.3134e-01	6.544e+01	2.0872e+01	3.1267e-03	6.54
9	8.81e-01	1.0072e+01	9.2727e+00	2.7086e-03	1.00
10	9.5111e-01	5.3348e-07	9.0174e+00	2.5165e-03	5.33



(a) Residual in log-scale

(b) Execution time in seconds, lin-scale

(c) Steps for convergence, log-scale

Fig. 2: Average residual (2a), execution time (2b) and steps (2c) for $\theta \in (0, \frac{pi}{2})$

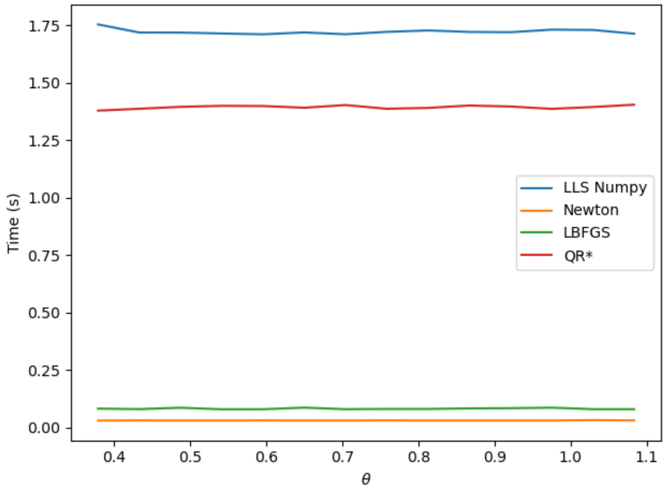
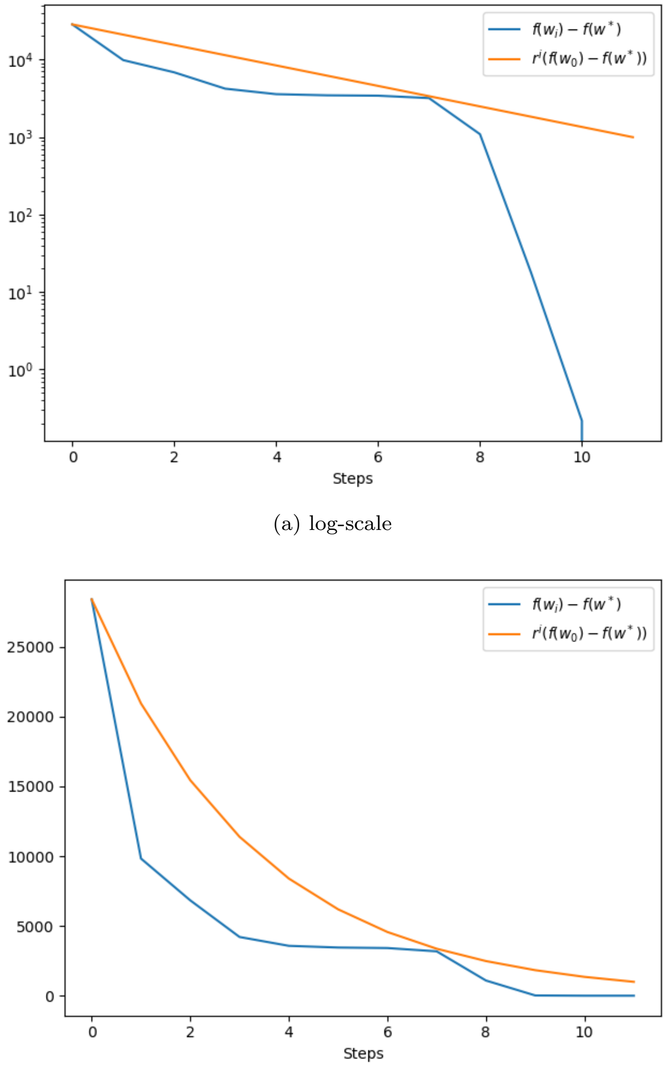


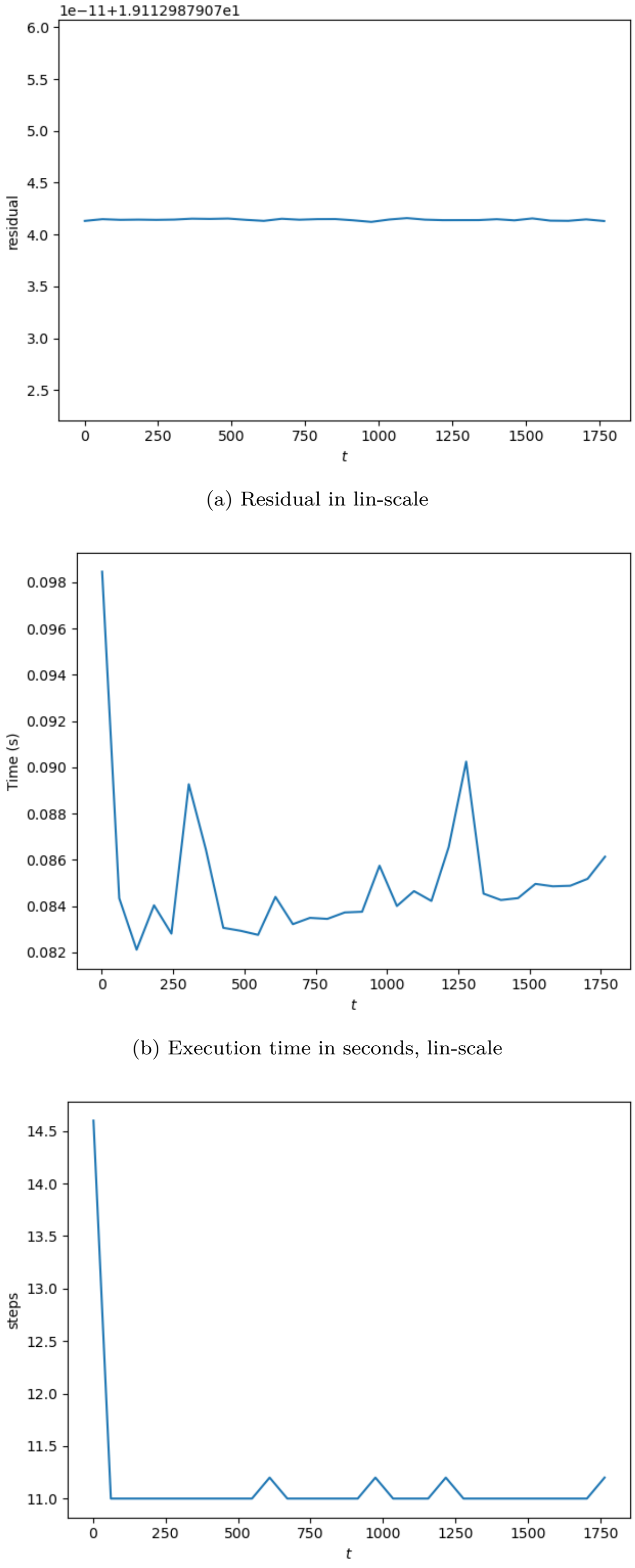
Fig. 3: Average execution time in for the LLS problem for $\theta \in (\frac{\pi}{8}, \frac{3\pi}{8})$



(a) log-scale

(b) lin-scale

Fig. 4: Average convergence over multiple runs of the L-BFGS algorithm with $r \approx 0.737$



(a) Residual in lin-scale

(b) Execution time in seconds, lin-scale

(c) Steps for convergence, lin-scale

Fig. 5: Average residual (5a), execution time (5b) and steps (5c) for L-BFGS with different memory $t \in (0, n)$