INFO-F524 : Technical Report

Safouan EHLALOUCH (514145), Ayoub AFIF (524527)

27th May 2024

1 Introduction

In this report, we will present and describe the development and analysis of a regression solver incorporating four iterative methods : the Iterative Shrinkage-Thresholding Algorithm (ISTA), the Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) and its restart variant, and the L-BFGS algorithm. This solver, developed in MATLAB, aims to extend a startup's capabilities in the domain of regression analysis. As emphasizing theoretical arguments only won't provide sufficiently convincing arguments in favor of FISTA, this project aims to numerically leverage the advantages of FISTA over ISTA. Additionally, we also compare a variant of FISTA which is FISTA with restart, and a quasi-Newton method which is L-BFGS. To do this comparison, we selected linear regression datasets of increasing size with different statistical proprieties artificially generated and from the real world.

2 Motivation for Algorithm Selection

This section aims to motivate the selection of the four iterative methods implemented.

2.1 Iterative Soft-Thresholding Algorithm

For the first time, we implemented the Iterative Soft-Thresholding Algorithm (ISTA). ISTA is a well-established proximal gradient method for solving l_1 regularized regression models (LASSO) and $l_1 l_2$ regularized models (elastic net). The main advantages of ISTA are its simplicity and effectiveness in promoting sparsity in the solution. This makes it particularly suitable for high-dimensional data where feature selection is essential. As both LASSO and elastic net introduce an l_1 norm penalty (or an $l_1 l_2$ in the case of elastic net), the function is therefore convex but not differentiable everywhere. ISTA can handle the LASSO and elastic net problems, even though these objective functions include non-differentiable terms. Therefore, ISTA is a good candidate for the startup.

ISTA works by iteratively updating the solution by performing a gradient descent step followed by a soft-thresholding operation to enforce sparsity. More precisely, the two steps can be detailed as follows.

- **Gradient Descent Step**: the algorithm computes a gradient descent step to minimize the least-squares objective function.
- **Proximal Operator Step** : proximal operator associated with the $l1$ norm to enforce sparsity. The proximal operator of a convex function finds the point that minimizes the sum of the value of the function and a term measuring the distance to a given point, scaled by a parameter. The proximal operator for the $l1$ norm is the soft-thresholding function:

$$\text{prox}_{\lambda}(y) = \text{sign}(y) \max(|y| - \lambda, 0)$$

Combining these two steps, we formally describe an iteration of ISTA as :

$$x_{k+1} = \text{prox}_{t_k \lambda}(x_k - t_k \nabla f(x_k))$$

where x_k is the current estimate of the regression coefficients at iteration k, t_k the step size, and $\nabla f(x_k)$ is the gradient of the least-squares loss function. For convex problems, it has been shown theoretically that ISTA has a convergence rate of $O\left(\frac{1}{k}\right)$ where k is the number of iterations.

2.1.1 Line search methods

Line search methods are essential for iterative methods to determine the optimal step size during iteration. We considered two strategies for computing the step size t_k at iteration k. In the next of this report, we compare the performance of the algorithms using both approaches for fixing the step size.

Backtracking line search: this iterative method is used to find an appropriate step size t. The goal is to ensure that each step reduces the objective function sufficiently while maintaining numerical stability and efficiency. Nevertheless, this method does not find the optimal step size t^* because it is an inexact line search method. It is important to note that computing t^* is usually computationally expensive and therefore using a backtracking line search method allows us to search for an acceptable step length. Formally, the backtracking line search algorithm used is described as follows.

Algorithm 1 Backtracking Line Search

Require: Initial step size $t_{init} > 0$, parameters c and α (contraction factor)
1: Initialize: Set $t \leftarrow t_{init}$
2: **while** $f(x_k + td_k) > f(x_k) + ct\nabla f(x_k)^T d_k$ **do**
3: Set $t \leftarrow \alpha t$
4: **end while**
5: **return** $t_k \leftarrow t$

Fixed Step Size: in contrast to the backtracking line search method, the fixed step size approach uses a constant value for t_k for all the iterations. The main advantages of this approach are the easy algorithmic implementation and can be computationally efficient. Nevertheless, the choice of this step size must be chosen carefully. On one hand, if t_k is too large, the algorithm may oscillate or diverge. On the other hand, if t_k is too small, the convergence might be slow and therefore a large number of iterations to reach a satisfactory solution are needed. When we used this approach, we fixed $t_k = \frac{1}{L}$ where L is the Lipschitz constant. This choice is common for computing a fixed step size because it ensures the convergence of the algorithm. It is explained by the fact that the Lipschitz constant provides an upper bound on the growth rate of the gradient. We estimated the Lipschitz constant by taking the maximum eigenvalue of the multiplication between A and A^T where A is the design matrix in linear regression. The rationale behind this choice is that the Lipschitz constant of its gradient is bounded by the maximum eigenvalue of the Hessian matrix of the objective function. In our case, we deal with a linear mapping and therefore the Hessian matrix is the multiplication between A and A^T.

2.2 Fast Iterative Shrinkage-Thresholding Algorithm

The Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) is an accelerated variant of ISTA that introduces a momentum term to reduce the number of iterations required to converge to an optimal solution. The algorithmic structure of FISTA is similar to the one of ISTA. It maintains the simplicity of ISTA while offering an improved convergence rate of $\mathrm{O}\left(\frac{1}{k^2}\right)$. Therefore, it achieves the optimal rate for first-order methods, i.e. a rate of $\mathrm{O}\left(\frac{1}{\sqrt{\epsilon}}\right)$. For this reason, we decided to implement this method to numerically demonstrate its improved convergence rate against ISTA.

To improve the convergence rate of ISTA, FISTA modifies the update rule of ISTA by adding a momentum term. At each iteration k, FISTA computes a linear combination of the current and previous iterates:

$$y_{k+1} = x_{k+1} + \frac{t_k - 1}{t_{k+1}}(x_{k+1} - x_k)$$

where x_{k+1} is the current iterate and x_k is the previous iterate. Then, FISTA performs a gradient descent step on the momentum term and applies the proximal operator to the updated momentum term to enforce sparsity. This combination of the momentum term and the proximal operator step improves the convergence of FISTA by taking larger steps toward the optimal solution. This acceleration is specifically useful in the initial iterations of the optimization process where FISTA outperforms ISTA in terms of convergence speed.

2.2.1 Line search method

We implemented only the fixed step size strategy (as described previously) for FISTA.

2.3 FISTA with restart

This method is a variant of FISTA. It modifies the standard FISTA algorithm to prevent the potential stalling of convergence. It can happen that the momentum term slow convergence (or oscillations) when the iterates get stuck in a suboptimal region. The idea of this algorithm is to periodically reset the momentum term to escape these problematic scenarios. Therefore, it ensures more consistent and reliable convergence behavior. It takes advantage of inertia and avoids oscillation. We implemented this algorithm to compare its performance with the standard FISTA algorithm.

To implement this algorithm, we simply added a restart condition in the standard FISTA algorithm. At each iteration, the algorithm checks the following condition to decide whether to restart the momentum term.

$$f(x_k) > f(x_{k-1}) \tag{1}$$

If this condition is true, we decided to restart the algorithm from a random point.

2.3.1 Line search method

As for the standard FISTA, we implemented only the fixed step size strategy.

2.4 Limited-memory Broyden–Fletcher–Goldfarb–Shanno Algorithm

The Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm belongs to the family of quasi-Newton methods. Quasi-Newton methods build up curvature information of the objective function to approximate the Hessian matrix. Computing this approximation improves the convergence rate compared to first-order methods. In traditional quasi-Newton methods such as the BFGS algorithm, the approximation of the inverse Hessian matrix must be stored and updated. This can be costly specifically when dealing with high-dimensional problems. To overcome this cost, L-BFGS maintains only a limited number of vectors that represent the approximation. This is called limited memory. It keeps track of a fixed number m of the most recent updates to the position vectors and the gradient vectors. Therefore, this algorithm is useful for solving large problems whose Hessian matrices cannot be computed at a reasonable cost or are not sparse. In terms of convergence rate, L-BFGS generally has a superlinear convergence rate and performs well for smooth convex problems.

The rationale behind the implementation of this algorithm is to present a comparison between a second-order method and first-order methods.

2.4.1 Line search method

As for ISTA, two line search methods were implemented for this algorithm: the fixed step size and the backtracking line search.

3 Design of the iterative solver

The iterative solver is designed to manage several optimization problems, including regularized regression models and other composite optimization tasks. As will show the results, we tested our solver on LASSO, Ridge, and elastic net. The iterative solver has been completely implemented from scratch in MATLAB. No subroutines or libraries have been used to design the solver. In this section, we detail the input and output specifications of our iterative solver.

3.1 Input/Output Specifications

Each algorithm has been implemented in a separate function. The inputs of all these functions are roughly the same. We can enumerate them as follows:

- A: design matrix that contains the features in column and the samples in rows of the regression problem.
- target vector b: vector containing the target values of the regression problem.

- *tolerance*: convergence criterion that involves monitoring the relative change in the objective function value. The algorithm stops when this change falls below a certain threshold given by the value of *tolerance*. The stopping criterion is therefore described in Equation 2.

$$|f(x_{k-1}) - f(x_k)| < tolerance \tag{2}$$

- *reg_model*: this parameter indicates which objective function the iterative solver must minimize. It takes three integer values corresponding to the objective function to select: 1 for LASSO regression, 2 for RIDGE regression, and 3 for elastic net regression. This allows the solver to handle several regularized regression models.

- *step_size_method*: integer parameter indicating the method for determining the step size. This parameter is only available for ISTA, and L-BFGS. FISTA and FISTA with restart do not have this parameter as input since they use only a fixed step size. It takes 1 for fixed size and 2 for backtracking line search.

All the functions output two variables :

- x_{old}: solution vector (estimate of the regression coefficients)

- *history*: variable of type `struct` that contains the iteration history. It contains information such as errors, timings, etc. All this information is essential for comparing the performances of the algorithms.

3.2 Computation of the proximal operator

The proximal operator is computed in the function *proximal_operator*. It maps a given vector y onto the solution of a proximal minimization problem involving a regularization term. It computes the proximal operator for LASSO, Ridge, and elastic net. Evaluating the proximal operator requires solving a minimization problem in general. Nevertheless, for the three functions considered, we have a closed-form expression that is immediate to evaluate.

1. **LASSO Regularization** (`reg_model = 1`):

$$\text{prox}_{\lambda_1\|\cdot\|_1}(y) = \text{shrink}(y, \lambda_1 t) = \text{sign}(y) \cdot \max(|y| - \lambda_1 t, 0)$$

   where λ_1 is the regularization parameter that controls the strength of the sparsity-inducing penalty. The sign(y) function preserves the sign of the components of y and the max function performs the soft-thresholding operation.

2. **Ridge Regularization** (`reg_model = 2`):

$$\text{prox}_{\lambda_2\|\cdot\|_2^2}(y) = \frac{y}{1 + 2\lambda_2 t}$$

   where λ_2 is the regularization parameter controlling the strength of the Ridge penalty. This operation shrinks each component of y towards zero by a factor proportional to $\lambda_2 t$.

3. **Elastic Net Regularization** (`reg_model = 3`):

$$\text{prox}_{\text{ElasticNet}}(y) = \text{prox}_{\lambda_1\|\cdot\|_1}(\text{prox}_{\lambda_2\|\cdot\|_2^2}(y))$$

   to compute the proximal operator of the elastic net regularization, the solver applies LASSO followed by Ridge regularization to the input vector y.

3.3 Line search methods

As explained before, two line search methods were designed. The first one is a simple fixed step size strategy and the second one is a backtracking line search strategy (*cf.* 2.1.1).

For the fixed step size, the step size is set at $t = \frac{1}{L}$ where L is the Lipschitz constant. The solver computes L by taking the gradient of the smooth part defined as:

$$\nabla f(x) = A^T(Ax - b) \tag{3}$$

From this gradient, the spectral normal i.e. the largest eigenvalue of the matrix A^TA is the Lipschitz constant L. This is because :

$$\|\nabla f(x) - \nabla f(y)\| = \|A^TA(x-y)\| \leq \|A^TA\|\|x-y\|$$

where $\|A^TA\|$ is the spectral norm of the matrix A^TA. Therefore, the solver computes the largest eigenvalue of this matrix using the `eig` function in MATLAB. It is important to notice that computing the eigenvalues of large matrices can be computationally expensive.

For the backtracking line search method, we simply implemented the described Algorithm 2.1.1. It reduces t by the contraction factor α until the Armijo condition is satisfied. It is important to note that the hyperparameters c and α used in this algorithm are essential for determining the step size t. However, we tested different combinations of values for these parameters but hyperparameter optimization is not the purpose of this project.

4 Regularized Regression Models

Our iterative solver handles regularized regression models and composite optimization problems as explained before. Regularized regression models introduce additional terms to the objective function to impose penalties on the model features. It helps to avoid overfitting of the model. We still have not introduced what these models represent. Therefore, this is the purpose of this section. Additionally, we present the type of composite optimization problems that can be solved by the solver.

Least Absolute Shrinkage and Selection Operator (LASSO)

The first regularized model handled by our iterative solver is the LASSO model. It minimizes the residual sum of squares subject to the sum of the absolute value of the coefficients being less than a fixed value. Its objective function is described in Equation 4.

$$f(x) = \min_x \left\{ \frac{1}{2}\|Ax-b\|_2^2 + \lambda_1\|x\|_1 \right\} \tag{4}$$

where λ_1 is the regularization parameter that controls the strength of the penalty. This model performs feature selection by shrinking some coefficients exactly to zero. Therefore, LASSO aims to find a sparse solution to a regression problem by minimizing a cost function with an l_1-norm penalty. This function is composed of a smooth, differentiable function which is the least square part, and a non-smooth function which is the regularization term. As described in a previous section, ISTA, and FISTA are suitable to tackle this type of function.

Ridge Regression

The second regularized model handled by our solver is the Ridge regression. It consists of adding a penalty term proportional to the squared magnitude of the coefficients. Ridge regression does not perform feature selection in the same way as LASSO. It shrinks all coefficients towards zero but none are exactly zero (except in rare cases). Its objective function is described in Equation 5.

$$f(x) = \min_x \left\{ \frac{1}{2}\|Ax-b\|_2^2 + \lambda_2\|x\|_2^2 \right\} \tag{5}$$

where λ_2 is the regularization parameter that controls the strength of the l_2 penalty. Unlike LASSO, this function is differentiable and smooth because the regularization term is a quadratic function and hence is smooth. This smoothness property allows for the use of standard gradient-based optimization techniques. No special considerations for non-smoothness are required to minimize this function.

Elastic net

The third regularized model considered by the solver is elastic net. This model combines the penalties of LASSO and Ridge regression. Therefore, it is defined as a convex combination of the l_1 and l_2 norms of the coefficients. As LASSO, elastic net performs feature selection but it tends to be less aggressive in reducing coefficients to exactly zero compared to LASSO. Its objective function is described in Equation 6.

$$f(x) = \min_x \left\{ \frac{1}{2}\|Ax - b\|_2^2 + \lambda_1\|x\|_1 + \lambda_2\|x\|_2^2 \right\} \tag{6}$$

where λ_1 and λ_2 are the regularization parameters that control the strength of the l_1 and l_2 penalties, respectively.

4.0.1 Composite Optimization Problems

Our solver handles composite optimization problems. These problems involve minimizing an objective function that consists of the sum of a smooth convex function and a non-smooth convex function. The general form of a composite optimization problem is described in Equation 7.

$$\min_x f(x) = g(x) + h(x) \tag{7}$$

where

- $g : \mathbb{R}^n \to \mathbb{R}$ is convex, differentiable, and admits a Lipschitz continuous gradient, i.e., ∇g is Lipschitz continuous with constant $L > 0$ over $\text{int}(\text{dom}(g))$, namely, $|\nabla g(x) - \nabla g(y)| \leq L\|x - y\|$ for all x, y.
- $h : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is closed, convex, possibly nondifferentiable, assumed with inexpensive prox-operator (prox_h).

4.1 Statistical properties of the datasets

For benchmark purposes and to evaluate the performance of our regression analysis, we used several datasets with different statistical properties. These properties give information about the characteristics of the data. It helps to assess how well the solver performs across different configurations of datasets. This section aims to outline the key statistical properties observed in the datasets used for regression analysis.

4.1.1 Artificial dataset generated

For the first time, to test our iterative solver, we generated an artificial dataset for a linear regression model. The rationale behind this choice is that generating an artificial dataset helps to simulate various scenarios and test the robustness of our iterative solver. We varied the size of the dataset i.e. the number of rows (samples) and the number of columns (features) of the design matrix A. It allowed us to evaluate the performance of the solver under different sample sizes and feature dimensions. However, for the next of this work, all the results presented related to this dataset were done using 100 samples and 100 features. We also varied the level of noise present in the linear relationship between the features and the target variable. This allows us to evaluate the robustness of the solver to noisy data. Additionally, this artificial dataset serves as a baseline performance measurement. It is a baseline for evaluating the performance of the solver. It allowed us to compare the convergence rate of ISTA and FISTA and therefore to validate our implementation. Consequently, it confirmed the correctness of the solver.

4.1.2 Profit dataset

The second dataset used for benchmarking is sourced from real-world datasets. This dataset is available on `Kaggle`. It aims to train a multiple linear regression model to predict the profit in the startup of a company. It contains 4 features which are: R&D (research and development), administration, marketing spending, and the location of the company (state). The target variable is the expected profit on the startup of a company. It contains 50 samples. This first real-world dataset is small in terms of both samples and features. However, it serves as a commendable initial step to assess the performance of our solver performance on data from real-world contexts. To have a better overview of the dataset, we computed the main statistical measures of the target variable, as shown in Table 3.

Mean	112012.63
Median	107978.19
Mode Variance	14681.4 1624588173.41
Range	14681.4 - 192261.83
Interquartile Range (IQR)	14681.4192261.83

Table 1: Statistical measures of the target variable profit

4.1.3 Cars dataset

The third dataset used for benchmark purposes is the Cars dataset. It has been designed to predict the weight of the car given some information about the car's performance. It contains the four following features: horsepower (HP), miles per gallon (MPG), volume (VOL), and speed (SP). The target variable is the weight (WP) of the car. This dataset is composed of 81 samples. Therefore, it contains more samples than the previous dataset. As for the previous dataset, we computed the main statistical measures of the target variable, as shown in Table 2.

Mean	32.41
Median	32.73
Mode Variance	15.71 56.14
Range	15.71 - 52.99
Interquartile Range (IQR)	7.80

Table 2: Statistical measures of the target variable weight

4.1.4 Student Performance dataset

The fourth dataset is the student Performance dataset. This dataset is designed to examine the factors influencing academic student performance. It consists of 10,000 student records (samples), with each record containing information about various predictors and a performance index. The four features are the following:

- Hours Studied: the total number of hours spent studying by each student.
- Previous Scores: the scores obtained by students in previous tests.
- Extracurricular Activities: whether the student participates in extracurricular activities (Yes or No).
- Sleep Hours: the average number of hours of sleep the student had per day.
- Sample Question Papers Practiced: the number of sample question papers the student practiced.

The target variable is the performance index. It is a measure of the overall performance of each student. The index ranges from 10 to 100, with higher values indicating better performance. The statistical overview of this target variable is presented in Table 3.

Mean	55.2248
Median	55
Mode Variance	67 369.12
Range	10 - 100
Interquartile Range (IQR)	31

Table 3: Statistical measures of the target variable performance index

5 Performance Analysis

In this section, we present the results of the numerical experiments done. The values of parameters used for the following plots are presented in Table 4. When different parameter values were used, we specifically mentioned it in the text.

λ_1	0.1
λ_2	0.1
tolerance	0.1
Seed	30

Table 4: Experiment's parameters

5.1 Artificial dataset

For the first time, we evaluated the solver on the artificial dataset generated with 100 samples and 100 features.

5.1.1 ISTA with fixed step size VS ISTA with backtracking line search

We will first compare the performance of the two line search methods in ISTA: fixed step size and backtracking line search. Figure 5.1.1 illustrates it.

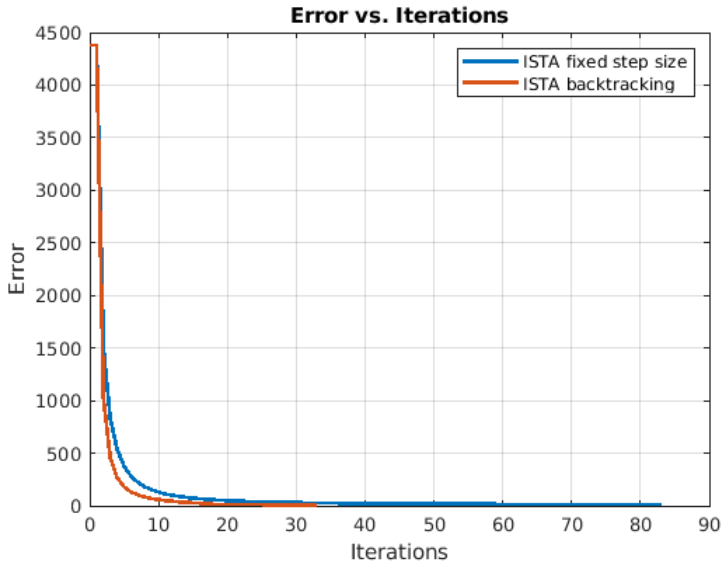

Figure 1: ISTA with fixed stepsize against ISTA with backtracking line search. ISTA with backtracking outperforms ISTA with a fixed step size.

In this first graph, we can see that ISTA backtracking converges more quickly towards the minimum error value at the beginning and finds the minimum error value after fewer iterations than ISTA fixed step size. ISTA backtracking finds the optimal value after 33 iterations and ISTA fixed step size after 83 iterations. The fixed step size line search method allows the algorithm to converge but not as quickly as the backtracking line search. This is because the stopping criterion of the backtracking line search ensures that the size of the step is not too big to satisfy the sufficient decrease condition but not too small to slow down the convergence.

The convergence of the algorithm with the backtracking line search is significantly better than with a fixed step size but it comes with a computation time cost. Computing the step size with a backtracking line search uses a loop that runs while a stopping criterion is not met.

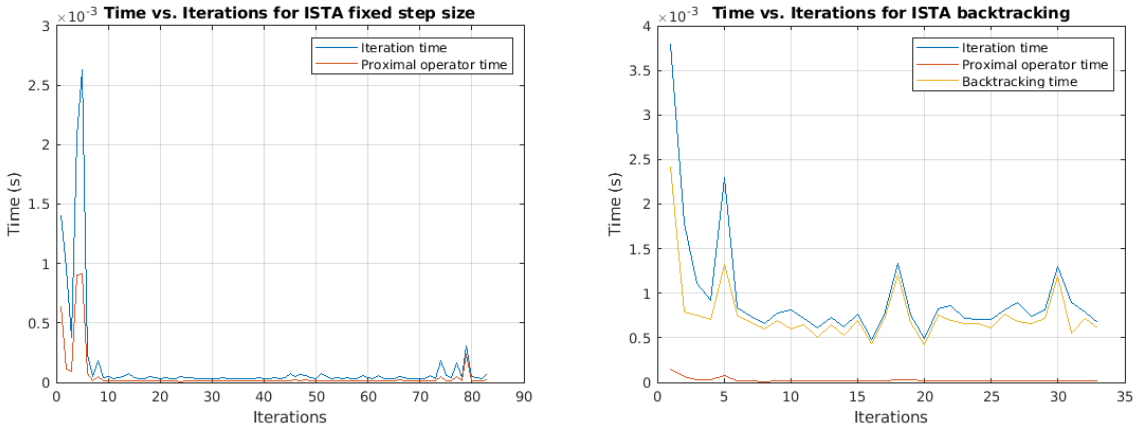

Figure 2: On the left, the computation time of each iteration of ISTA with a fixed step size. On the right is the computation time of each iteration of ISTA with a backtracking line search.

In these two graphs, we can see which part of the algorithm is the most time-consuming between the computation of the proximal operator and the computation of the step size compared with the whole iteration time. We can observe that the backtracking computation time is the part that is the most consuming for the backtracking line search algorithm. In comparison, the computation of the fixed step size is done in one instruction. We can also observe that the proximal operator computation is the most time-consuming part for the fixed step size algorithm while it's almost insignificant for the backtracking line search algorithm. The whole algorithm took 0.0216s to compute for the fixed step size algorithm and 0.0415s to compute for the backtracking line search algorithm.

ISTA with fixed step size takes a lot more iteration before finding the optimal solution for the objective function than ISTA with backtracking line search but it is almost twice as fast because the computation of the step size at each iteration of the algorithm is very time-consuming.

Two parameters could be adapted to make the backtracking line search more efficient in the backtracking algorithm 2.1.1: the parameter c and the contraction factor α. The parameter c is a value between 0 and 1 and controls how much the objective function has to decrease at each iteration. We can then adapt the value of c to make the objective function decrease more or less and thus control how many loops the backtracking will go through. The α parameter is a value between 0 and 1 and controls how much the step size will decrease at each loop of the backtracking algorithm. If we increase the value of α, the algorithm will be more precise in the choice of the step size but will take more loop steps at each iteration before satisfying the stopping criterion. We can thus sacrifice some precision to make the algorithm use fewer loop steps.

5.1.2 FISTA

We first analyze how the error decreases for each iteration. Figure 3 shows the evolution of the error of FISTA.

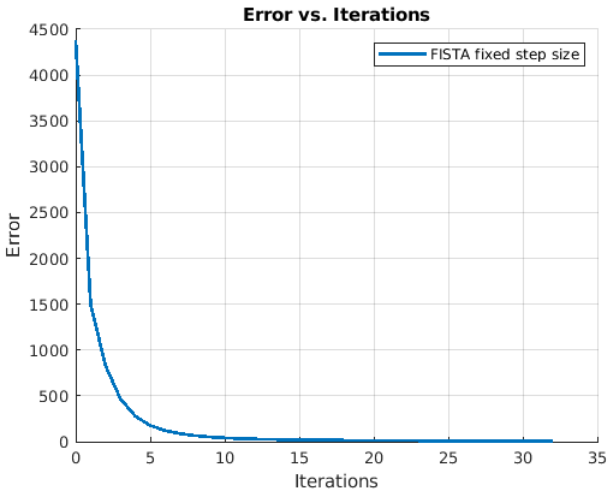

Figure 3: The decrease of the error for each iteration of FISTA.

We can see that FISTA converges very well at the beginning. FISTA takes 32 iterations and 0.0058s to find the optimal solution which makes it better than ISTA in terms of iterations and total computation time. ISTA with backtracking line search and FISTA with fixed step size have almost the same number of iterations but FISTA is seven times faster in terms of computation time. Here, we can see which part of the algorithm is the most time-consuming:

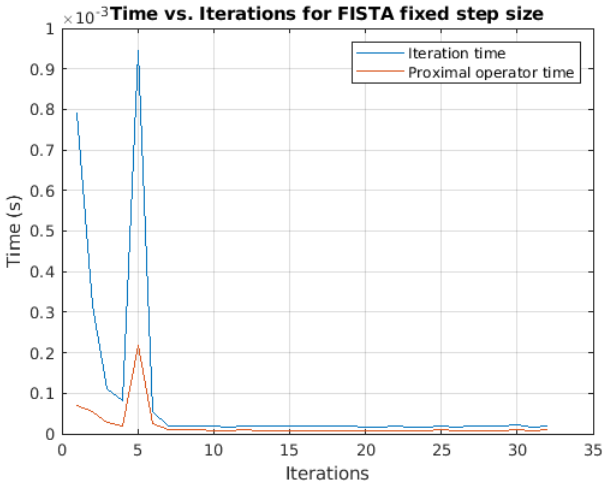

Figure 4: Computation time of each iteration of FISTA and the computation time of the proximal operator for each iteration.

We can observe that the computation of the proximal operator does not represent the biggest proportion of the computation time for each iteration and the step size is computed with one instruction. This can be explained by the fact that we know the closed-form expression of the proximal operator. Therefore, it is immediate to evaluate the different functions.

We also implemented FISTA with a restart mechanism. We do the same numerical experiment as for FISTA to analyze the evolution of the error for each iteration, as shown in Figure 5.

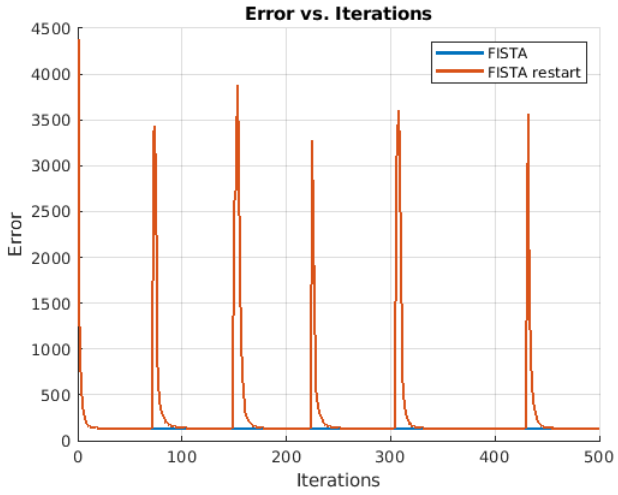

Figure 5: Comparison of the decrease of the error for each iteration for FISTA and FISTA restart.

We can observe in Figure 5 that the two algorithms have the same performances but the error with FISTA restart gets really big instantly at some points. This is when a random solution is generated after the algorithm stops improving. In our case, the restart is useless because FISTA already finds the optimal solution without the restart mechanism. Additionally, the objective function has only one minimum. Therefore, there is no chance that the algorithm get stuck in suboptimal regions.

5.1.3 L-BFGS fixed step size VS L-BFGS backtracking

We will now compare the performance of L-BFGS with the two line search methods, as shown in Figure 6.

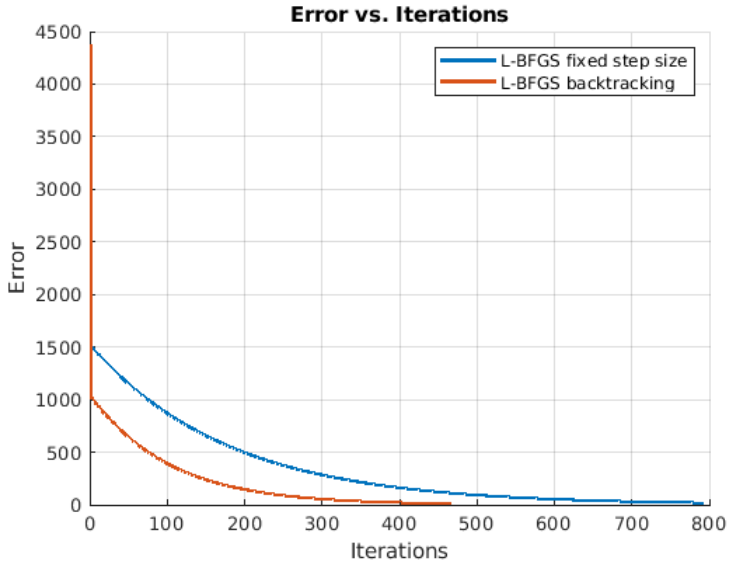

Figure 6: Comparison of the decrease of the error for each iteration for L-BFGS with fixed step size and L-BFGS with backtracking.

The two algorithms converge very quickly at the first iteration and very slowly just after that. The performance in terms of iterations and total computation time is really bad compared to ISTA and FISTA. L-BFGS with fixed step size took 794 iterations and 3.1639s to find an optimal solution and

L-BFGS with backtracking line search took 468 iterations and 2.3987s. We can therefore conclude that the algorithm using the backtracking algorithm converges better than the one using a fixed step size.

The two-loop recursion is the part that is the most time-consuming of the algorithm, as we can see in Figure 7.

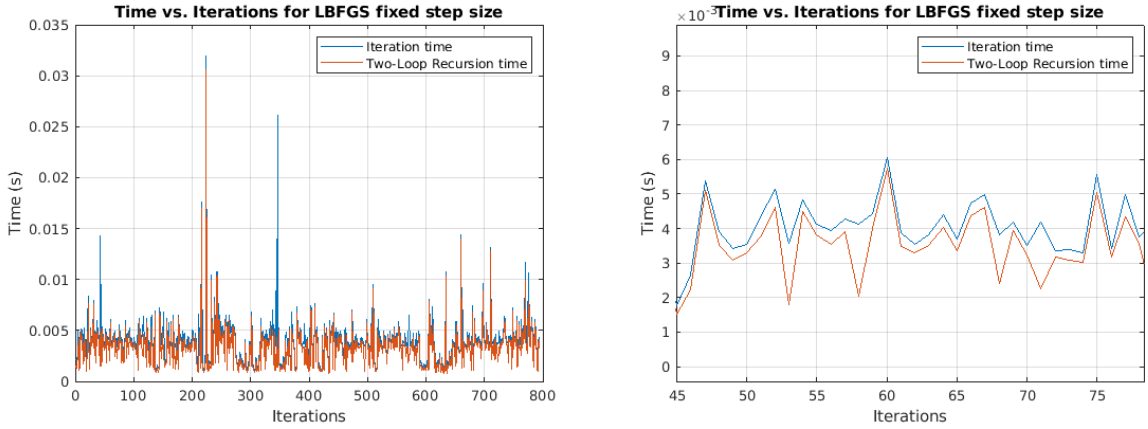

Figure 7: On the left, the computation time of each iteration of L-BFGS with a fixed step size. On the right, it is a zoom on the graph.

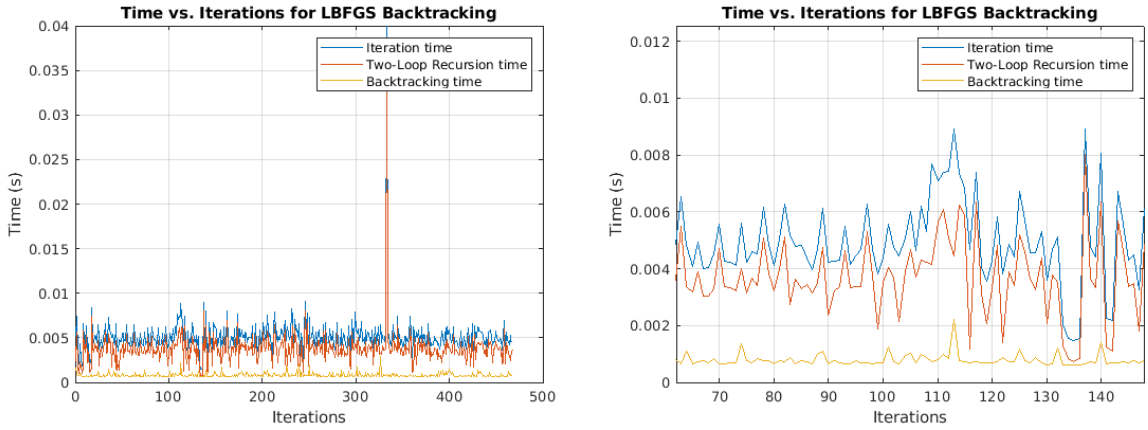

Figure 8: On the left, the computation time of each iteration of L-BFGS with backtracking line search. On the right, it is a zoom on the graph.

We can also observe that the computation of the step size with the backtracking line search, which was the most time-consuming in the case of ISTA, is now really low compared to the whole computation time of the iteration. This is illustrated in Figure 8.

5.1.4 Comparison of the algorithms

In this section, we will compare the performance of the algorithms with each other.

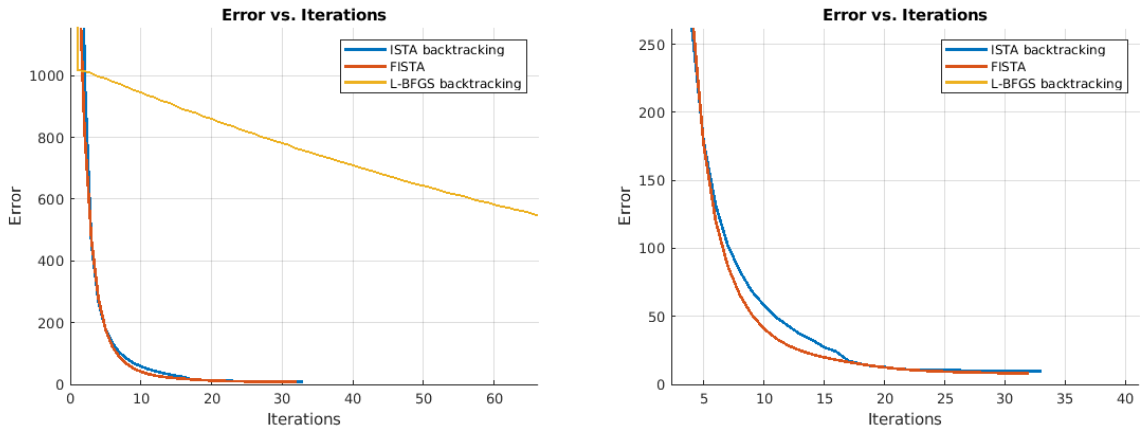

Figure 9: On the right, comparison of the decrease of the error for each iteration for ISTA with backtracking line search, FISTA, and L-BFGS with backtracking line search. On the left, a zoom on the first graph.

In Figure 9, we only used the version of each algorithm that had the best performances in terms of iterations. We can see that L-BFGS with backtracking is by far the worst of the three. As mentioned before, ISTA with backtracking and FISTA with fixed step size have almost similar performances in terms of iterations but in terms of computation time, FISTA is a lot faster. We can compare the computation time of all the algorithms in the following graphs.

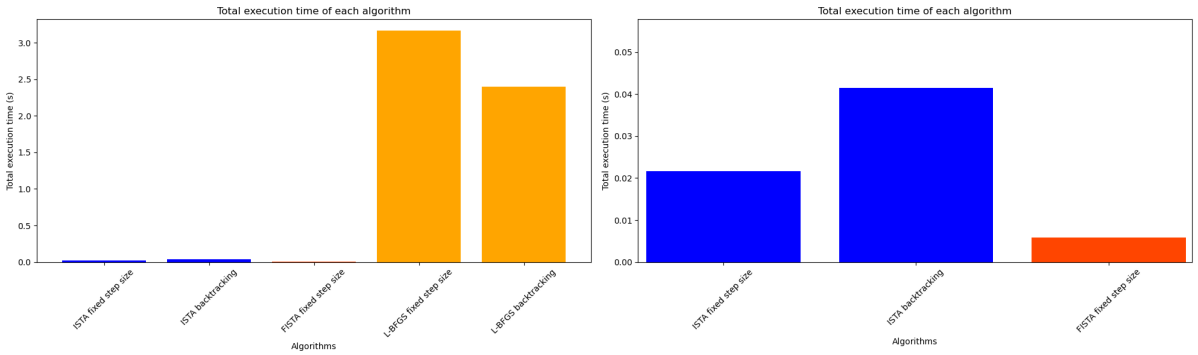

Figure 10: On the right, comparison of the total computation time of each algorithm. On the right, a focus on ISTA and FISTA.

We can see that L-BFGS is by far the worst in terms of computation time. We can also observe a very low computation time for FISTA compared to ISTA.

5.1.5 Different size of datasets

In this section, we compare the performance of the different algorithms (fixed step size) with datasets of increasing sizes for the LASSO problem. There are three sizes of datasets: 100 samples and 100 features, 200 samples and 200 features, and 300 samples and 300 features.

The following graph illustrates the performance of ISTA.

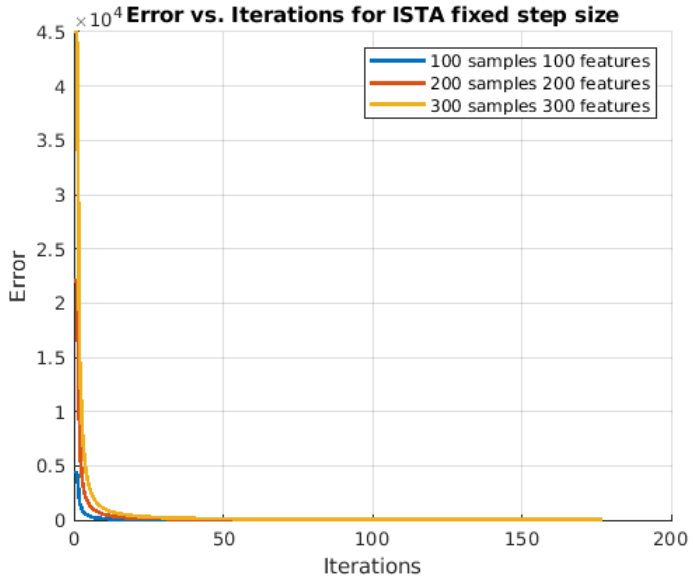

Figure 11: Comparison of the decrease of the error for each iteration for ISTA with datasets of increasing sizes.

The first and smallest dataset took 82 iterations and 0.0156s to find the optimal solution. The second dataset took 146 iterations and 0.0188s. Finally, the last and largest dataset took 178 iterations and 0.0252s. Additionally, the following graph illustrates the performance of FISTA.

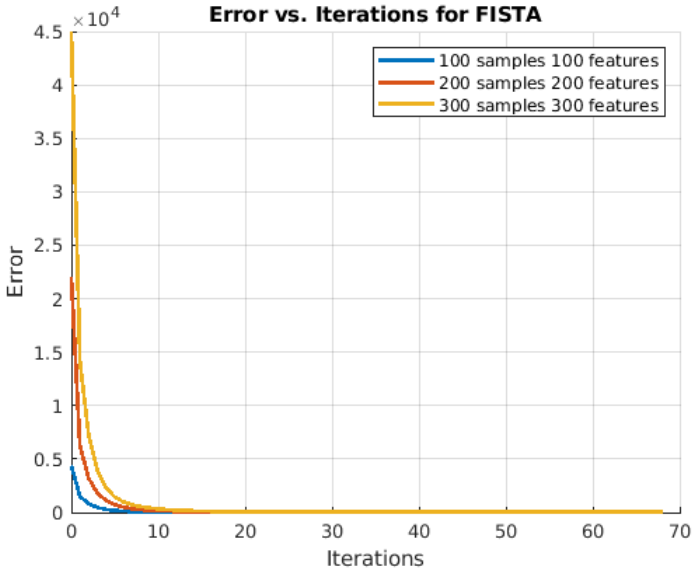

Figure 12: Comparison of the decrease of the error for each iteration for FISTA with datasets of increasing sizes.

The first dataset took 32 iterations and 0.0048s to find the optimal solution. The second dataset took 51 iterations and 0.0048s. Finally, the last dataset took 68 iterations and 0.0068s.

The last algorithm to analyze is L-BFGS. The results are illustrated in Figure

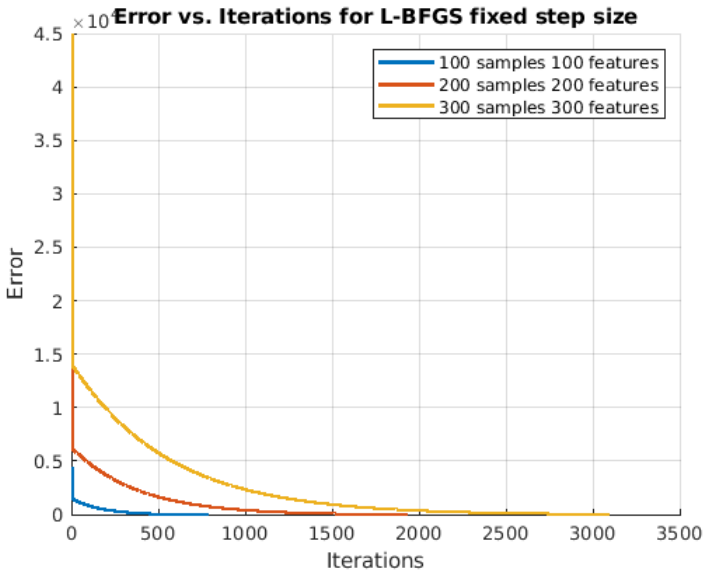

Figure 13: Comparison of the decrease of the error for each iteration for L-BFGS with datasets of increasing sizes.

The first dataset took 794 iterations and 2.5779s to find the optimal solution. The second dataset took 1939 iterations and 7.0028s. And the last dataset took 3099 iterations and 11.6160s.

Overall, FISTA had again the best performance in terms of iterations and computation time for every dataset size. We can also observe that the more we increase the size of the dataset, the more iterations and computation time the algorithm will need to find the optimal solution.

5.1.6 Different regularized regression models

We can also compare the difference in performance of our different algorithms (fixed step size) for all the regression models (LASSO, Ridge, elastic net) to seek for any variation in performance. The l_1 and l_2 regularized parameters were both set to 2 in these experiments.

In the case of ISTA, here is our graph:

Figure 14: Comparison of the decrease of the error for each iteration for ISTA with LASSO, Ridge, and Elastic Net.

The optimal solution was found after 80 iterations for LASSO, 67 iterations for Ridge, and 65 iterations for elastic net.

Then the graph of FISTA:

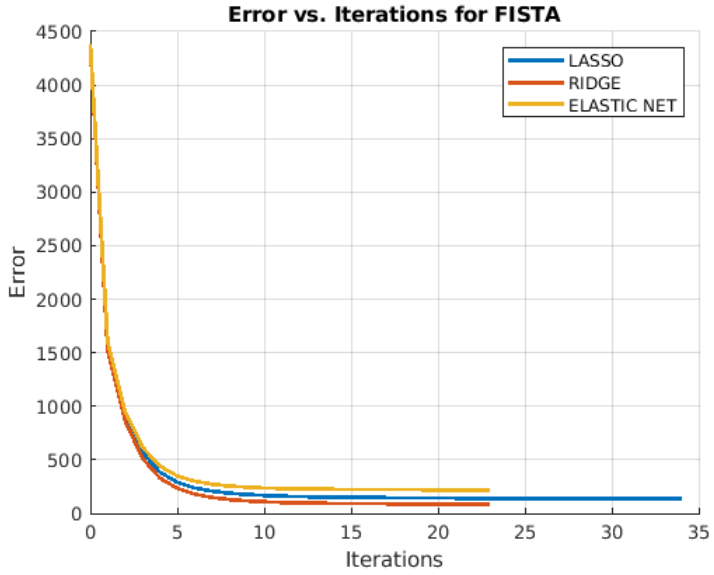

Figure 15: Comparison of the decrease of the error for each iteration for FISTA with LASSO, Ridge, and Elastic Net.

The optimal solution was found after 34 iterations for LASSO, 23 iterations for Ridge, and 23 iterations for elastic net.

And finally, the graph for L-BFGS:

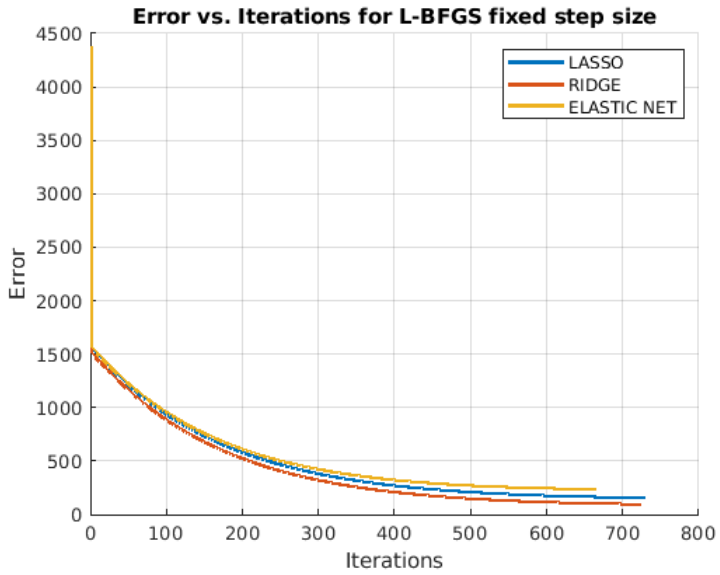

Figure 16: Comparison of the decrease of the error for each iteration for L-BFGS with LASSO, Ridge, and Elastic Net.

The optimal solution was found after 731 iterations for LASSO, 726 iterations for Ridge, and 667 iterations for elastic net.

We can observe that for every algorithm, LASSO is the regression model that takes the most iterations before finding the optimal solution, followed by Ridge and then Elastic Net. Our hypothesis about this behavior is that, since LASSO produces a more sparse solution because it sets the coefficient to exactly zero, it can take significantly more time to converge. Using fewer variables in the dataset to find the optimal solution can lead to some instability.

5.1.7 L-BFGS memory size

One important parameter in the L-BFGS algorithm is the memory size m. It is usually in the range [3,20] but we tested the algorithm with three values of m: 3, 16, and 32. Here is the graph with LASSO and the l_1 regularized parameter set to 2:

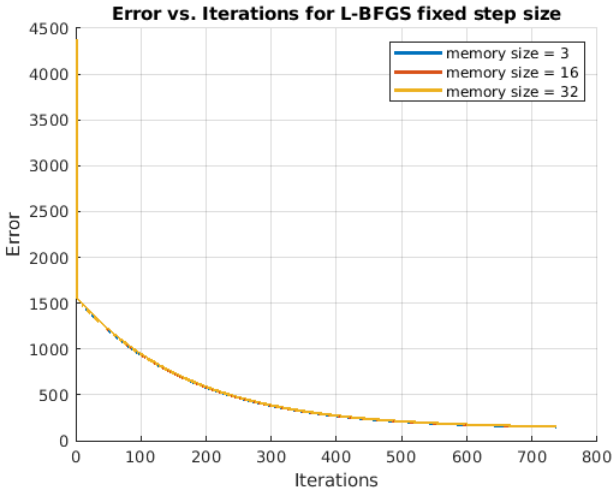

Figure 17: Comparison of the decrease of the error for each iteration for L-BFGS with increasing memory size.

The algorithm with $m = 3$ found the optimal solution after 739 iterations and both the algorithm

with $m = 16$ and $m = 32$ found the best solution after 738 iterations. The algorithm with a larger memory found the best solution with fewer iterations but it is not a significant difference.

5.1.8 Influence of the regularized parameter on L-BFGS

While doing our experiments, we witnessed an unexpected behavior of L-BFGS when dealing with really big values of the regularized parameter. In the next graph, we used L-BFGS (fixed step size) with the LASSO regression model and three values of the l_1 regularized parameter: 0.1, 10, and 50.

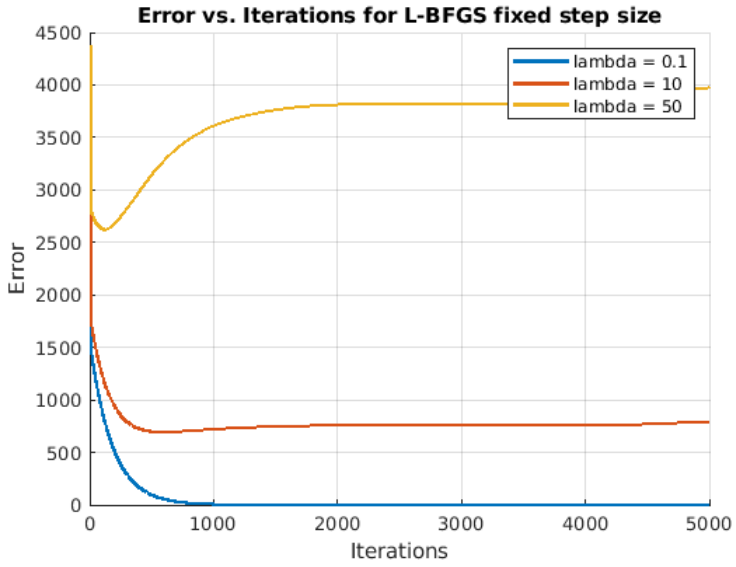

Figure 18: Comparison of the decrease of the error for each iteration for L-BFGS with increasing lambda value.

As we can observe in the graph, when the value of the l_1 regularized parameter gets larger, the algorithm tends to diverge. For the same problem with the l_1 regularized parameter equal to 50, FISTA found a best solution with an error of 2180. We can see that, in the beginning, L-BFGS was going in the right direction before diverging. We tried using a Python solver (`scipy.optimize.minimize`) that uses the L-BFGS-B algorithm and it could not resolve the problem either. That made us think that the behavior of L-BFGS was not due to an incorrect implementation of our code. There are many reasons why L-BFGS could behave this way but our main hypothesis is that the weakness of L-BFGS is that it works less well on ill-conditioned problems. With a very big value of the l_1 regularized parameter, a small variation in the input data can lead to a big variation in the solution which means that our problem is ill-conditioned and it could be the reason for this unexpected behavior.

5.2 Datasets from real-world

We also tried our algorithms on real datasets that we found on `Kaggle`. Here are the results for ISTA (backtracking line search), FISTA, and L-BFGS (backtracking line search) with the l_1 regularized parameter equal to 2:

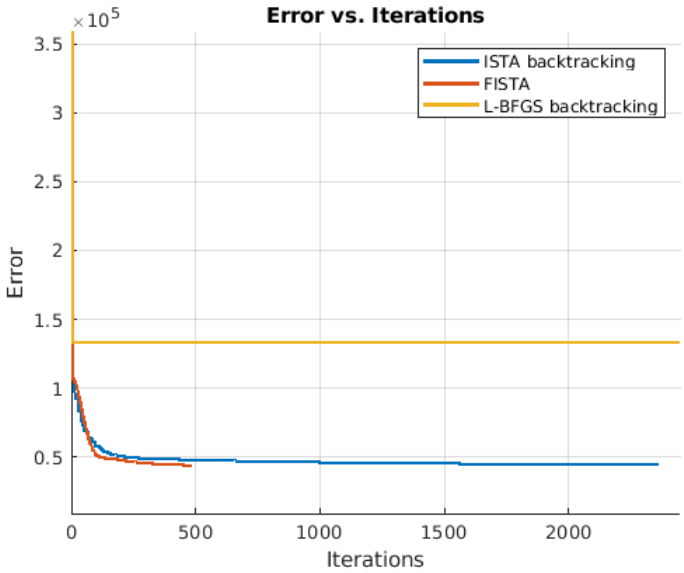

Figure 19: Comparison of the best algorithms on the `Student Performance` dataset

In this dataset, ISTA found the best solution after 2360 iterations, FISTA after 487 iterations, and L-BFGS did not find any solution.

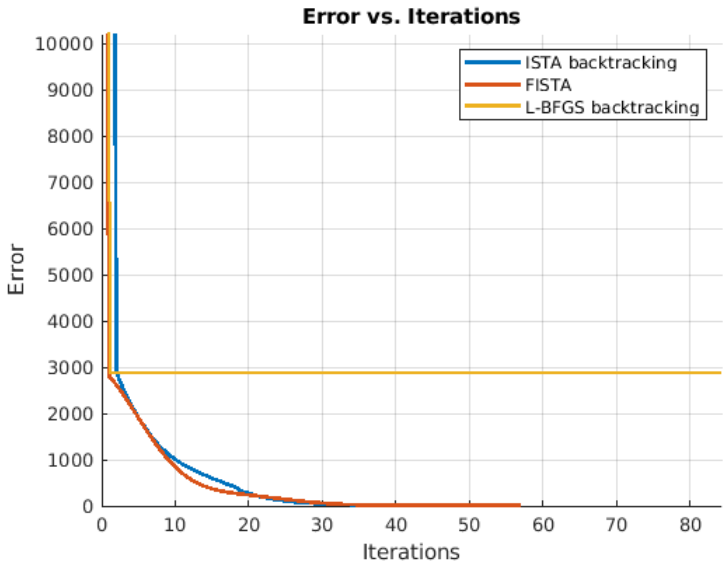

Figure 20: Comparison of the best algorithms on the `Cars` dataset

In this dataset, ISTA found the best solution after 54 iterations, FISTA after 58 iterations, and L-BFGS did not find any solution.

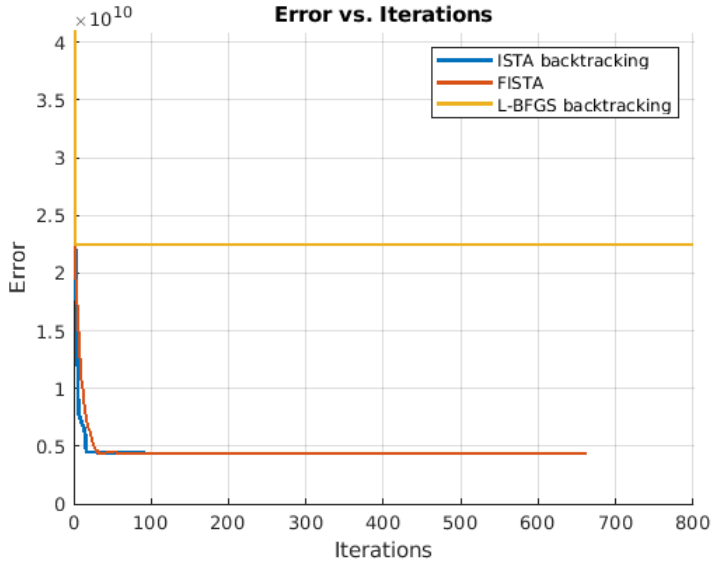

Figure 21: Comparison of the best algorithms on the `Profit` dataset

In this dataset, ISTA found the best solution after 92 iterations, FISTA after 664 iterations, and L-BFGS did not find any solution. We can also observe that in two of the three datasets, ISTA found the solution with fewer iterations than FISTA. L-BFGS could not find any solution for the three datasets and the error was not decreasing at all.

6 Conclusion

To conclude this technical report, our comprehensive analysis encompassed both theoretical understanding and numerical implementation, using a variety of datasets to benchmark the algorithms. We can conclude that the FISTA showed a significantly better overall convergence rate than ISTA. This is consistent with the theoretical arguments in terms of convergence rate. Therefore, it makes a valuable algorithm for regularized regression problems such as LASSO, Ridge, and Elastic net. Additionally, we can conclude that L-BFGS has the worst performance in terms of convergence rate among all the considered algorithms. Concerning the line search methods, the fixed step size approach provided a straightforward and reliable method but the backtracking line search contributed to overall faster convergence rates. This improvement came at the cost of additional computational overhead. The findings of this analysis open up several avenues for future research and development such as exploring hybrid approaches to get better results.

A Code source

The source code is available on our public GitHub repository, which can be accessed at :
https://github.com/sehlalou/Continuous.git.