University of Novi Sad
Faculty of Sciences
Department of Mathematics and Informatics

Student
Isidora Vuković

Application of a Stochastic Spectral Gradient Method on Machine Learning Problems

Master Thesis

Mentor
Dr Nataša Krklec Jerinkić

Novi Sad, 2024

I would like to express my sincere gratitude to my mentor Dr Nataša Krklec Jerinkić for her advice, help, support, and suggestions during the research and writing of this thesis. It was a real pleasure working with her. I would also like to thank the other members of the board, Dr Sanja Rapajić and Dr Dušan Jakovetić. Additionally, I extend my gratitude to all other professors and teaching assistants who selflessly helped me gain knowledge during my studies.
My deepest gratitude goes to my mother Slobodanka, my father Jovan, my sisters, Nevena and Tijana, and my brother Manojlo, who have been my greatest support in everything I do and who shaped my character with their upbringing and behavior.

Table of contents

1 Introduction **4**

2 Machine learning **6**
2.1 Numerical Optimization in Machine Learning 8

3 Problem Description **11**
3.1 Line Search Method 11
3.2 Subsampling 13

4 Spectral gradient method **16**

5 The algorithm **20**
5.1 Convergence analysis 22

6 Numerical results **24**
6.1 Changing the number of internal iterations m 25
6.2 Changing the size of subsample S 28
6.3 Thresholds γ_{min} and γ_{max} 32
6.4 Comparison of SGD and SLiSeS 36

7 Conclusions **38**

The Python code **39**

Bibliography **46**

Biography **48**

1 Introduction

The problem of minimizing finite sums (a sum of a finite number of so-called local loss functions) often arises in various fields such as machine learning, statistics, and economics, making its solution crucial. Consequently, constructing algorithms for solving finite sum problems and continuously improving them remains a subject of ongoing research in the scientific community.

In response to this challenge, numerous techniques have been developed, theoretically suitable for addressing this problem. Among the earliest and most well-known is the gradient descent method. The idea behind this method is straightforward: iteratively move in the direction of the negative gradient of the function to decrease its value. However, its main drawback is slow convergence. To achieve faster convergence rates, various variations and improvements of the gradient descent method have emerged.

One of these advanced techniques is the Newton's method. In addition to first-order information, it requires finding the Hessian matrix in each iteration. However, this process has proven to be costly, as computing and manipulating the Hessian matrix can be demanding, particularly for functions with a large number of variables. To overcome these challenges, methods that approximate the Hessian matrix in different ways have been developed. Among them is the spectral gradient method.

The spectral gradient method has proven to be extremely efficient in classical (deterministic) optimization. However, in practice, most classical methods are not applicable because the optimization process costs too much. Therefore, increasing attention is being paid to stochastic methods. In the paper [11], the authors proposed and analyzed the Subsampled Line Search Spectral Gradient Method for Finite Sums - SLiSeS, where the advantages of the spectral gradient method are utilized within the framework of stochastic optimization. In stochastic optimization, instead of using the entire dataset, a randomly selected data sample is used for each iteration to reduce computational costs and speed up the optimization process. It is important to note that stochasticity introduces a certain level of noise into the optimization process. For this reason, the authors of the SLiSeS algorithm proposed retaining the same data sample for several iterations before selecting a new sample. This allows for better exploration of the objective function's structure and contributes to the efficiency of the mentioned method.

The aim of this study will be to investigate the behavior of the SLiSeS algorithm in various scenarios. The focus will be on analyzing the impact of different parameters on the algorithm's performance. Additionally, we will explore how the SLiSeS algorithm behaves in situations with different types of data or different types of optimization problems. The goal is to gain a deeper understanding of how SLiSeS operates in different situations and how it can be most effectively applied in practice. This research will help us identify best practices and recommendations for using the SLiSeS algorithm in real-world applications.

The rest of this thesis is organized as follows. In Chapter 2, we will explain the concept of machine learning and focus on the importance of numerical optimization in the context of machine learning. In Chapter 3, we will present a description of the optimization problem that is considered in this thesis. After that, we will explain the Line Search method and Stochastic Gradient Descent (SGD) as elements of the SLiSeS algorithm. In Chapter 4, we will explain in detail the spectral gradient method and present some of its characteristics. In Chapter 5, we will present the SLiSeS algorithm itself with the corresponding line search technique and explain their steps in detail. We will also state the conditions that guarantee the convergence of this algorithm to a stationary point. Numerical experiments are given in Chapter 6. Finally, in Chapter 7, we will provide some concluding remarks.

2 Machine learning

Machine learning (ML), a discipline of artificial intelligence (AI), enables machines to learn automatically from data and past experiences. This allows them to identify patterns and make predictions with minimal human intervention. It has a wide range of applications, including image and speech recognition, natural language processing, recommendation systems, autonomous vehicles, medical diagnosis, fraud detection, and much more. Machine learning involves showing a large volume of data to a machine so that it can learn and make predictions, find patterns, or classify data.

There are three types of machine learning: supervised, unsupervised, and reinforcement learning. Each of these has distinct advantages in different situations, depending on the nature of the problem and the desired output.

Supervised learning is used when we have labeled input data and target output values (labels), and the goal is to learn a model that can predict or classify new input data based on those labeled instances. To achieve this, the machine is trained on a set of example inputs and corresponding outputs. This type of learning is very common and is used to solve problems such as classification (where we try to classify input data into discrete classes) and regression (where we try to predict continuous output values).

In **unsupervised learning** the machine is not provided labeled examples or previous patterns on which to base the analysis of the input data. The machine must uncover patterns and draw inferences by itself, without having the correct answers. It will classify or cluster data by discovering the similarity of features on its own. This approach enables a deeper understanding of the data and potentially the discovery of new information that could be useful for various analyses and decision-making.

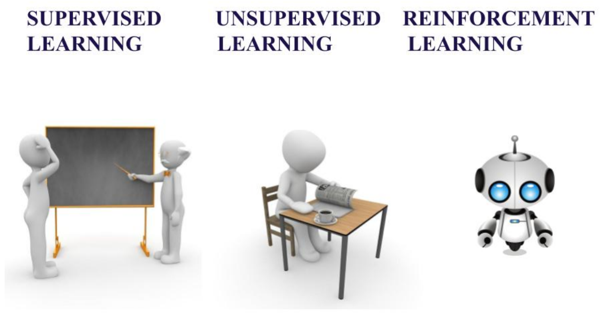

Figure 1: Supervised, Unsupervised and Reinforcement learning, Source: [1]

Reinforcement learning differs from both supervised and unsupervised learning primarily because it is used to take actions to achieve certain goals in dynamic environments and continuously improves its model based on feedback from experiences. It learns through trial and error - from the consequences of his actions and new choices. As the action is taken, the success of the outcome is evaluated, receiving a positive or negative rating. The algorithm strives to obtain positive results and the model is trained on continuous feedback. A conceptual example of this would be a self-driving car that would get a positive rating for moving from one location to another without crashing.

In this thesis, the focus will be on supervised learning, so here follows a brief explanation of how this type of learning works.

The first step is to collect relevant data so that the machine learning model can find the correct patterns. The quality of the data fed into the machine will determine how accurate the model is; thus, inaccurate or outdated data can lead to inaccurate outcomes or predictions that lack relevance. High-quality data is characterized by its relevance, minimal instances of missing or duplicated values, and a comprehensive representation of different subcategories or classes. Once the relevant data is available, it should be prepared, which can be achieved by, for example, handling missing values, "normalizing data," and "encoding categorical variables." The next step is splitting the cleaned data into two sets - a training set and a testing set. The training set is used to train the machine learning model, while the testing set is used to assess the model's accuracy after training. Then an appropriate algorithm should be chosen depending on the nature of the problem and the characteristics of the data. After choosing the algorithm, the model needs to be trained, which is the most important step in machine learning. During training, training data is fed into a machine learning model to find patterns and make predictions. This learning process involves adjusting model parameters to minimize errors between the predicted outcomes and the actual data, ultimately enhancing the model's accuracy and its ability to make precise predictions. After training the model, it is necessary to evaluate its performance. This is done by testing the model's ability on data it has not seen before, usually using a test set previously separated from the training data. Various metrics are used to measure how well the model performs, such as accuracy, precision, recall, and root mean square error.

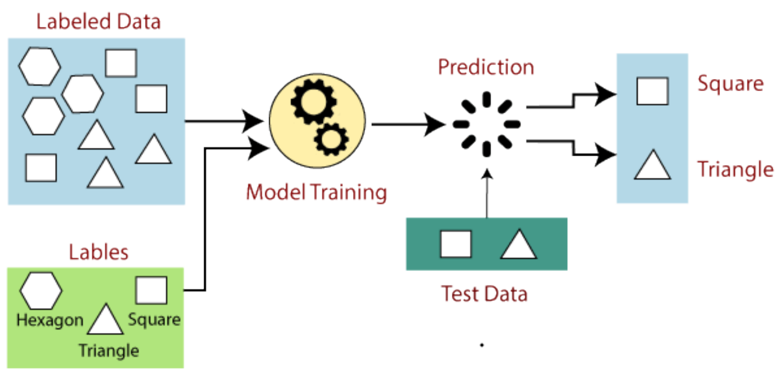

Figure 2: Steps of Supervised Machine Learning, Souce: [2]

The choice of evaluation metric depends on the specific problem and the desired outcome. Once the model has been created and evaluated, it should be checked whether its accuracy can be improved in any way. This is achieved by adjusting the "hyperparameters" present in the model. Hyperparameters are variables in the model that are generally chosen by the programmer. At a certain value of the hyperparameter, the accuracy reaches its maximum. Setting hyperparameters involves identifying these values. In the end, the model can be applied to unseen data for accurate predictions.

2.1 Numerical Optimization in Machine Learning

At the very core of machine learning lies numerical optimization, which involves the process of finding the best possible solution from a set of feasible options, taking into account various constraints and objectives. This optimization process plays a key role in adjusting the parameters of machine learning models, improving their performance and allowing them to draw general conclusions from the available data, so that they can successfully apply their knowledge to new situations they have not been exposed to before.
In numerical optimization the following problem is considered:

$$\min_{x \in S} f(x), \tag{2.1}$$

where $f : D \to \mathbb{R}$ and $D, S \subseteq \mathbb{R}^n$. Vector x is called the decision variable, and its dimension n represents the dimension of the problem. The function f is called the objective function, and D represents its domain. S is called the feasible set and represents the constraints of the optimization problem (2.1). In the case when $S = \mathbb{R}^n$, the problem (2.1) is said to be an unconstrained optimization problem. On the other hand, if S is a true subset of $\mathbb{R}^n$, then the problem (2.1) is called a constrained optimization problem, and S is usually stated as follows:

$$S = \{x \in \mathbb{R}^n \mid h(x) = 0, g(x) \leq 0\}, \tag{2.2}$$

where $h : \mathbb{R}^n \to \mathbb{R}^m$ represents equality constraints, and $g : \mathbb{R}^n \to \mathbb{R}^p$ represents inequality constraints. Both of these constraints are explicit constraints, while implicit constraints are represented by the domain D.

Solving the optimization problem means finding the best feasible decision variable - the one that minimizes the objective function on the feasible set. There are two types of solutions to the problem (2.1), global and local solutions, and their definitions follow below.

Definition 1. A point x^* is a global solution of the problem (2.1) if $f(x^*) \leq f(x)$ for every $x \in S$. If $f(x^*) < f(x)$ for every $x \in S$, $x \neq x^*$, then x^* is considered a strict global solution.

Definition 2. A point x^* is a local solution of the problem (2.1) if there exists $\varepsilon > 0$ such that $f(x^*) \leq f(x)$ for every $x \in S$ such that $\|x - x^*\| \leq \varepsilon$. If $f(x^*) < f(x)$ for every $x \in S$, $x \neq x^*$, such that $\|x - x^*\| \leq \varepsilon$, then x^* is termed a strict local solution.

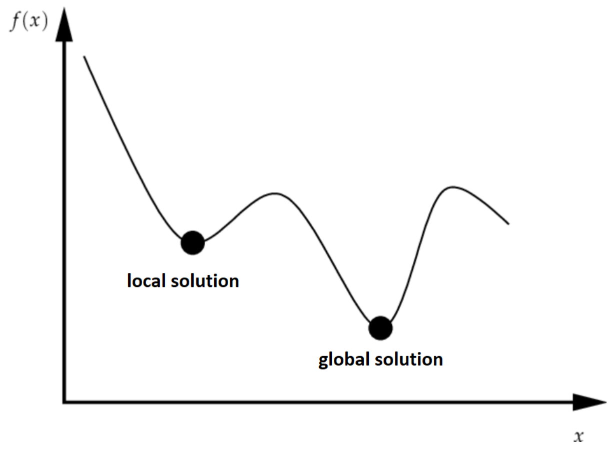

Figure 3: Local and global solution to the problem (2.1), Source: [3]

Finding a global solution can be very challenging, making local solutions particularly relevant in nonlinear optimization. This thesis focuses on analyzing the behavior of algorithms designed for solving unconstrained optimization problems:

$$\min_{x \in \mathbb{R}^n} f(x), \tag{2.3}$$

where $f : \mathbb{R}^n \to \mathbb{R}$. Here, we state the first and second necessary conditions, as well as a sufficient condition, for the local solution of these problems.

Theorem 1. *Suppose that $f \in C^1(\mathbb{R}^n)$. If x^* is a local solution of* (2.3)*, then $\nabla f(x^*) = 0$.*

Theorem 2. *Suppose that $f \in C^2(\mathbb{R}^n)$. If x^* is a local solution of* (2.3)*,then*

a) $\nabla f(x^*) = 0$*;*

b) $\nabla^2 f(x^*) \succeq 0$.

Theorem 3. *Suppose that $f \in C^2(\mathbb{R}^n)$. If*

1. $\nabla f(x^*) = 0$ *and*
2. $\nabla^2 f(x^*) \succ 0$

then x^* is a strict local solution of (2.3).

There are many numerical methods for finding a local solution to the problem (2.3), such as gradient descent, Newton's method, Quasi-Newton method, and many others. Each of these methods has its advantages and disadvantages, often evaluated in terms of speed of convergence and efficiency in working with different types of functions. For example, some methods feature a fast convergence rate, but may require additional information about the function, which can be expensive or difficult to obtain. This leads scientists to research and develop new approaches to improve existing methods and overcome their shortcomings. Additionally, researchers experiment with combining different techniques or adapting existing methods to specific problem characteristics. This approach contributes to the development of more efficient and robust numerical algorithms capable of handling increasingly complex challenges in the field of machine learning and optimization. Advances in this area offer new opportunities to efficiently solve complex machine learning problems, thereby contributing to the development of advanced models and algorithms used in various applications.

3 Problem Description

In machine learning, a large number of problems involve computing the approximate minimizer of a finite sum of local loss functions over a large number of training examples. These local loss functions estimate how much the actual value for a given data point differs from the value predicted by the model. The basic idea of this problem is to find a set of parameters or variables that minimize the total or global loss, usually represented as the sum of these local loss functions. The global loss function quantifies how well the model predictions align with the true target values for the entire dataset, and minimizing this global loss is essential for training models to achieve accurate predictions and good generalization. However, practical problems often introduce additional complexities. For example, local loss functions in practice are often nonconvex, which complicates the optimization problem because the problem may have multiple local minima.

We will consider the average of a finite number of possible non-convex smooth functions:

$$\min_{x \in \mathbb{R}^n} f(x) := \frac{1}{N} \sum_{i=1}^{N} f_i(x), \tag{3.1}$$

where each $f_i : \mathbb{R}^n \to \mathbb{R}$ for $i \in \{1, ..., N\}$ is bounded from below and f_{low} denotes the objective function's lower bound.

In the paper [11], the authors proposed the use of the Subsampled Line Search Spectral Gradient Method to solve the problem (3.1). The reasons for creating this algorithm, as well as its components, will be explained below.

3.1 Line Search Method

The line search method is an iterative approach to find a local minimum of a continuously differentiable multidimensional nonlinear function. Its basic concept derives from the observation that that if the point x is not a stationary point for the objective function f, i.e. $\nabla f(x) \neq 0$, as indicated by Theorem 1, it implies that x is not a minimizer of function f. Consequently, there is a vector $d \in \mathbb{R}^n$ known as the descent direction, defined below, such that $f(x + \alpha d) < f(x)$ for some scalar $\alpha > 0$, referred to as the step size. The descent direction d indicates a decrease in the function's value, and the step size α determines the extent of the movement along this direction.

Definition 3. Consider a point x such $\nabla f(x) \neq 0$. A direction d is called a descent direction for f at the point x if there exists $\alpha > 0$ such that

$$f(x + \alpha d) < f(x). \tag{3.2}$$

Below is stated a characterization of descent directions that is often used when f is continuously differentiable.

Theorem 4. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$, $f \in C^1(\mathbb{R})$, and $x \in \mathbb{R}^n$ is such that $\nabla f(x) \neq 0$. Moreover, suppose that the direction d satisfies the following inequality*

$$\nabla^T f(x)d < 0. \tag{3.3}$$

Then, there exists $\hat{\alpha}$ such that $f(x + \alpha d) < f(x)$ for all $\alpha \in (0, \hat{\alpha}]$.

A model algorithm of the line search method is given below.

Algorithm 1 Line Search Method

```
S0 Input parameters: $x^0 \in \mathbb{R}^n$.
S1 Initialization: $k = 0$.
S2 Stopping criterion: If $\nabla f(x^k) = 0$ STOP. Otherwise go to Step 3.
S3 Search direction: Choose $d^k$ such that $\nabla^T f(x^k)d^k < 0$.
S4 Step size: Find $\alpha_k > 0$ such that $f(x^k + \alpha_k d^k) < f(x^k)$.
S5 Update: Set $x^{k+1} = x^k + \alpha_k d^k$, $k = k + 1$ and go to S2.
```

However, the sequence $\{x^k\}$ generated by Algorithm 1 does not converge to a minimizer of f in every case. The algorithm will stop only if it encounters a stationary point of the function f, otherwise it generates a sequence $\{x^k\}$ for which it only holds $f(x^{k+1}) < f(x)$. In order to achieve convergence, it is necessary to introduce certain constraints. To avoid too small steps and directions that are nearly orthogonal to the gradient, the following conditions on direction d are imposed respectively:

$$||d^k|| \geq \sigma||\nabla f(x^k)||, \tag{3.4}$$

$$\nabla^T f(x^k)d^k \leq -\theta||\nabla^T f(x^k)||||d^k||, \tag{3.5}$$

where $\sigma > 0$, $\theta \in (0, 1]$. Also, in order to avoid too large steps, the following condition is imposed on the step size α_k:

$$f(x^k + \alpha_k d^k) \leq f(x^k) + \eta\alpha_k \nabla^T f(x^k)d^k, \tag{3.6}$$

where $\eta \in (0, 1)$. Condition (3.6) is often called the Armijo condition or the sufficient decrease condition. The following theorem states the conditions under which we can be certain that there exists an α satisfying (3.6).

Theorem 5. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$, $f \in C^1(\mathbb{R}^n)$, and $\nabla^T f(x^k)d^k < 0$. Moreover, assume that the function f is bounded from below on the line $\{x^k + \alpha d^k | \alpha > 0\}$. Then, there exists $\bar{\alpha} > 0$ such that the Armijo condition holds for all $\alpha \in (0, \bar{\alpha}]$.*

Now, the following theorem, under certain conditions, provides for the global convergence of the improved line search method.

Theorem 6. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$, $f \in C^1(\mathbb{R}^n)$ and f is bounded from below. Moreover, assume that the sequence of search directions $\{d^k\}_{k\in\mathbb{N}}$ is bounded. Then, either Algorithm 1, where for d^k and α_k the conditions 3.4, 3.5 and 3.6 hold, terminates after a finite number of iterations $\bar{k}$ at the stationary point $x^{\bar{k}}$ or every accumulation point of the sequence $\{x^k\}_{k\in\mathbb{N}}$ is a stationary point of the function f.*

The descent direction can be computed by various methods, among them is the gradient descent method, where $d^k = -\nabla f(x^k)$. This direction satisfies both inequalities (3.4), and (3.5) with $\sigma = 1$, and $\theta = 1$. The basic idea of this method is to take repeated steps in the opposite direction of the gradient of the function at the current point because this is the direction of steepest descent. However, despite its simplicity, the gradient descent method performs poorly (the rate of convergence is at most linear in general). Therefore, to achieve a better convergence rate, methods have been developed that, in addition to the gradient, also use second-order information. These will be discussed in the next chapter.

3.2 Subsampling

In many machine learning problems, the need to compute an approximate minimizer of problem (3.1) arises when dealing with a large set of training examples, which often exhibit a large amount of redundancy. In those cases, it is almost mandatory to employ stochastic iterative methods that update the prediction model based on a relatively small randomly chosen subset (or sample) of the training data. One such powerful method is Stochastic Gradient Descent (SGD), a variant of the Gradient Descent algorithm specifically designed for optimizing machine learning models. SGD efficiently addresses the computational inefficiency of traditional Gradient Descent methods when dealing with large datasets in machine learning projects. In SGD, instead of using the entire dataset for each iteration, only a single random training example (or a small batch) is selected to calculate the gradient and update the model parameters. This random selection introduces randomness into the optimization process, hence the term "stochastic" in stochastic Gradient Descent.

In SGD, the path the algorithm uses to reach the minimum is usually noisier than that of a typical Gradient Descent algorithm. The reason for this is that the algorithm randomly chooses only one sample from the full sample to estimate the gradient of the objective function for each iteration. However, the manner in which the algorithm moves is inconsequential, as long as it reaches a minimum with significantly shorter training time.

In the images below, we can see the paths of a typical Gradient Descent and the Stochastic Gradient Descent method (SGD). These paths provide insight into the differences between the two algorithms and how they move towards the minimum of the function.

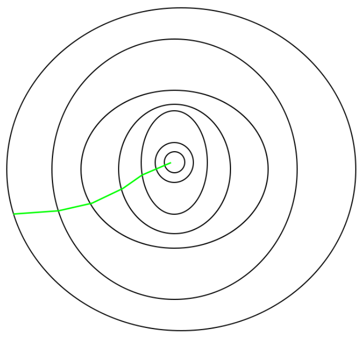

Figure 4: Gradient Descent Optimization Path, Source: [4]

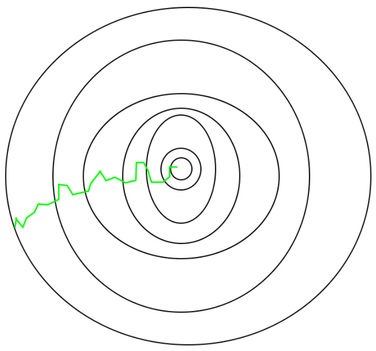

Figure 5: Stochastic Gradient Descent Optimization Path, Source: [4]

As we mentioned before, SGD is generally noisier than typical Gradient Descent, so it usually requires a higher number of iterations to reach the minimum, given the stochasticity in its descent. However, SGD is computationally much cheaper than typical Gradient Descent, although reaching the minimum requires more iterations than typical Gradient Descent.

Now that we have explained the reasons for introducing stochasticity into numerical methods, in the following, we will mathematically define SGD.

Let us denote the full sample by $\mathcal{N} = 1, 2, ..., N$ and, independently of the applied iterative method, let us denote the randomly chosen subsample at iteration k by $\mathcal{N}_k \subseteq \mathcal{N}$ where $|\mathcal{N}_k| = S \ll N$.The function estimator obtained by averaging the functions f_i in $\mathcal{N}_k$ is given by

$$f_{\mathcal{N}_k}(x) = \frac{1}{S} \sum_{i \in \mathcal{N}_k} f_i(x), \tag{3.7}$$

and the associated gradient estimator is

$$\nabla f_{\mathcal{N}_k}(x) = \frac{1}{S} \sum_{i \in \mathcal{N}_k} \nabla f_i(x). \tag{3.8}$$

For solving (3.1), stochastic gradient (SG) algorithms are of the form

$$x_{k+1} = x_k - \gamma_k \nabla f_{\mathcal{N}_k}(x_k). \tag{3.9}$$

However, in the case when the choice of the step length $\gamma_k > 0$ is standard (constant or decreasing), SG methods must perform a large number of iterations to observe an adequate reduction in the objective function.

In order to speed up the convergence rate of the iterative process towards the local minimum of problem (3.1), the Stochastic Gradient method has been enriched by incorporating spectral step lengths. The spectral method, which will be comprehensively explained in the next chapter, and its variants for the general unconstrained minimization problem (full sample), are low-cost gradient methods that have proved to be very effective in practice for large-scale optimization. However, all schemes that utilized this combination of spectral steps with the SG method shared a common feature of traditional options for changing the random subset $\mathcal{N}_k$ (with constant or dynamically increasing sizes) at each iteration. Subsequently, the authors of the SLiSeS algorithm proposed retaining the same subset for several iterations before subsampling again. As demonstrated in their numerical experiments, this alteration in the algorithm results in a significant improvement in the practical behavior of the method.

4 Spectral gradient method

In order to achieve the fastest possible convergence to the local minimum of the objective function, various numerical methods have been developed. These methods usually require some information about the Hessian in addition to using the gradient of the objective function. Since the Hessian represents the second-order derivative of the objective function, methods that use this information are commonly called second-order methods. Among them, Newton's method is of particular importance.

Therefore, the basic idea of Newton's method is to iteratively update the current solution, taking into account both the gradient and the curvature of the objective function. To better understand how Newton's method works, let us consider an iteration x^k such that $\nabla f(x^k) \neq 0$. In an ideal scenario, the goal is to progress to the next iteration x^{k+1} such that $\nabla f(x^{k+1}) = 0$. Denote $d^k = x^{k+1} - x^k$. Then, using the Taylor expansion yields the following approximation:

$$\nabla f(x^{k+1}) \approx \nabla f(x^k) + \nabla^2 f(x^k) d^k. \tag{4.1}$$

So, instead of searching for x^{k+1} such that the left-hand side is equal to zero, attention is shifted to finding d^k such that the right-hand side equals zero, i.e.

$$\nabla f(x^k) + \nabla^2 f(x^k) d^k = 0. \tag{4.2}$$

Equation (4.2) is called Newton's equation, and the vector d that satisfies it is commonly referred to as the Newton step or the Newton direction. It is a solution of the system of linear equations:

$$\nabla^2 f(x^k) d^k = -\nabla f(x^k). \tag{4.3}$$

This step is not unique in general, but if the Hessian matrix $\nabla^2 f(x^k)$ is non-singular, then the Newton step can be expressed as:

$$d^k = -(\nabla^2 f(x^k))^{-1} \nabla f(x^k). \tag{4.4}$$

As can be noted, the Newton method requires computing the Hessian and then solving the system of linear equations at each iteration. That can be computationally very expensive and challenging. In order to avoid these problems, Quasi-Newton methods were developed.

Quasi-Newton methods aim to approximate the Newton direction without explicitly calculating the Hessian matrix, which is achieved as follows. The quasi-Newton direction d^k should satisfy the following equation:

$$B_k d^k = -\nabla f(x^k). \tag{4.5}$$

where B_k denotes the approximation of the Hessian matrix. Then, assume that we have an approximation B_k and that we performed the iteration to obtain x^{k+1}.

Subsequently, we need to update the Hessian approximation B_{k+1}.To ensure a valid update, it is essential that the new approximation B_{k+1} satisfies the secant equation:

$$B_{k+1}s_k = y_k, \tag{4.6}$$

where:

$$s_k = x^{k+1} - x^k, \tag{4.7}$$

$$y_k = \nabla f(x^{k+1}) - \nabla f(x^k). \tag{4.8}$$

The secant method, which was originally developed for solving the problem of finding zeros (roots) of a function with one variable, became the basis for quasi-Newton methods for solving optimization problems of multidimensional functions. By avoiding the direct calculation of the Hessian matrix, these methods strike a balance between the computational efficiency of first-order methods and the convergence advantage of second-order methods.

In the **spectral gradient method** (originally introduced by Barzilai and Borwein [10], the search is for a matrix B_{k+1} with a very simple structure, which satisfies (4.6). More precisely, the condition is set that the matrix B_{k+1} is of the form:

$$B_{k+1} = \sigma_{k+1}I, \tag{4.9}$$

where $\sigma_{k+1} \in \mathbb{R}$, so then equation (4.6) becomes:

$$\sigma_{k+1}s_k = y_k. \tag{4.10}$$

In general, this equation has no solutions. However, accepting the least-squares solution that minimizes $||\sigma s_k - y_k||_2^2$, it is obtained that the solution is $\sigma_{k+1} = \frac{s_k^T y_k}{s_k^T s_k}$. Therefore, in the spectral gradient method, the descent direction is obtained by $d^{k+1} = -\sigma_{k+1}^{-1}\nabla f(x^{k+1})$, where

$$\gamma_{k+1}^{LONG} = \frac{1}{\sigma_{k+1}} = \frac{s_k^T s_k}{s_k^T y_k} \tag{4.11}$$

gives an approximation of the inverse Hessian called the **long BB step size**.

The secant equation can also be stated as $H_{k+1}y_k = s_k$, and in that case, the corresponding approximation of the inverse Hessian turns out to be

$$\gamma_{k+1}^{SHORT} = \frac{s_k^T y_k}{y_k^T y_k}, \tag{4.12}$$

and it is called the **short BB step size**.

It holds that:

$$\gamma_{k+1}^{SHORT} = \frac{s_k^T y_k}{y_k^T y_k} \leq \frac{||s_k||||y_k||}{||y_k||^2} \leq \frac{||s_k||^2}{||s_k||||y_k||} \leq \frac{s_k^T s_k}{s_k^T y_k} = \gamma_{k+1}^{LONG}. \tag{4.13}$$

When the objective function is a quadratic function:

$$f = \frac{1}{2}x^T A x^T + b^T x + c \tag{4.14}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n,n}$ is a symmetric positive definite (SPD) matrix, long and short BB step sizes become:

$$\gamma_k^{LONG} = \frac{\nabla f(x^{k-1})^T \nabla f(x^{k-1})}{\nabla f(x^{k-1})^T A \nabla f(x^{k-1})}. \tag{4.15}$$

$$\gamma_k^{SHORT} = \frac{\nabla f(x^{k-1})^T A \nabla f(x^{k-1})}{\nabla f(x^{k-1})^T A^2 \nabla f(x^{k-1})}. \tag{4.16}$$

respectively. So, $\frac{1}{\gamma_k^{LONG}}$ is a Rayleigh quotient of A by vector s_{k-1}, and $\frac{1}{\gamma_k^{SHORT}}$ is a Rayleigh quotient of A by vector $\sqrt{A}s_{k-1}$. Based on this and the min-max theorem, it follows that $\frac{1}{\gamma_k^{LONG}}$ and $\frac{1}{\gamma_k^{SHORT}}$ are in between the minimum λ_{min} and the maximum eigenvalue λ_{max} of the Hessian A:

$$0 \le \lambda_{min} \le \frac{1}{\gamma_k^{SHORT}} \le \frac{1}{\gamma_k^{LONG}} \le \lambda_{max}. \tag{4.17}$$

Note that the long BB step size (4.15) used for defining x^{k+1} is the one used in the optimal Cauchy steepest descent method for defining the step at iteration k. Therefore, the spectral gradient method computes, at each iteration, the step that minimizes the quadratic objective function along the negative gradient direction but, instead of using this step at the k-th iteration, saves the step to be used in the next iteration. It can be shown that the method converges for the strictly convex quadratic functions in the general case (for every n).

In the general case, by the Mean-Value Theorem of integral calculus, one has:

$$y^k = \left[\int_0^1 \nabla^{\mathbf{2}} f(x^k + ts^k)\, dt\right] s^k. \tag{4.18}$$

Therefore, $\frac{1}{\gamma_k^{LONG}}$ defined in (4.11) and $\frac{1}{\gamma_k^{SHORT}}$ defined in (4.12) are Rayleigh quotients relative to the average Hessian matrix $\int_0^1 \nabla^{\mathbf{2}} f(x^k + ts^k)\, dt$. Also, it follows that $\lambda_{min} \le \frac{1}{\gamma_k^{SHORT}} \le \frac{1}{\gamma_k^{LONG}} \le \lambda_{max}$, where λ_{min} and λ_{max} are respectively the minimum and the maximum eigenvalue of the average Hessian. This observation provides motivation for the term **spectral method**.

For minimizing general (not necessarily quadratic) functions, the spectral gradient method shows (highly) non-monotonic behavior during the convergence process, but it is significant that this method and its modifications show global convergence under mild assumptions. Also, as can be noted that in general, the long and short BB steps are not always greater than zero. In the case of a minimization task, and if these steps turn out to be negative or extremely large, it is advisable to choose an alternative positive value to ensure a stable and efficient optimization.

The spectral method and its variants, for a general (full sample) unconstrained minimization problem, are inexpensive gradient methods that have proven to be very efficient in practice for large-scale optimization. They have received much attention in the last three decades, including theoretical understanding, extensions and adaptations for different scenarios (unconstrained and constrained) and for some specific applications.

5 The algorithm

The basic idea of the authors of the SLiSeS algorithm was to utilize the advantages of the spectral gradient method within the framework of stochastic optimization. To reduce the impact of noise caused by the stochastic factor, the authors suggested that the same data sample be stored for several iterations before selecting a new sample. In this way, the algorithm is allowed to better explore the structure of the objective function, thereby increasing the probability of finding a convergence direction with less variation caused by noise.

In this chapter, SLiSeS (Subsampled Line Search Spectral Gradient Method for Finite Sums) will be described in Algorithm 2, while the line search procedure will be described in Algorithm 3. Also, the steps involved in these two algorithms will be explained in detail.

Algorithm 2 SLiSeS (Subsampled Line Search Spectral Gradient Method)

```
S0 Initialization: $x_0 \in \mathbb{R}^n$, $\eta \in (0,1)$, $S \in 0,1,...,N$, $m \in \mathbb{N}$, $0 < \gamma_{min} \leq 1 \leq \gamma_{max} < \infty$, $\{t_k\} \in \mathbb{R}^{\infty}_{+}$ such that $\sum_k t_k \leq \bar{t} < \infty$, and $\mathcal{N}_0 \subseteq \mathcal{N}$, $|\mathcal{N}_0| = S$. Set $k = 0$.
S1 Sampling: If $mod(k,m) = 0$ choose $\mathcal{N}_k \subseteq \mathcal{N}$ such that $|\mathcal{N}_k| = S$. Else, set $\mathcal{N}_k = \mathcal{N}_{k-1}$.
S2: Compute $g_k = \nabla f_{\mathcal{N}_k}(x_k)$.
S3 Spectral coefficient: If $mod(k,m) = 0$ and $m > 0$ set $c_k = \frac{1}{||g_k||}$. Else, $s_{k-1} = x_k - x_{k-1}$, $y_{k-1} = g_k - g_{k-1}$ and set $c_k = \frac{||s_{k-1}||^2}{(s_{k-1}^T y_{k-1})}$.
S4: Set $\gamma_k = \min\{\gamma_{max}, \max\{\gamma_{min}, c_k\}\}/k$.
S5 Search direction: Set $d_k = -\gamma_k \nabla f_{\mathcal{N}_k}(x_k)$.
S6 Step size: Find $\alpha_k \in (0,1]$ such that
    $$f_{\mathcal{N}_k}(x_k + \alpha_k d_k) \leq f_{\mathcal{N}_k}(x_k) + \eta \alpha_k \nabla^T f_{\mathcal{N}_k}(x_k) d_k + t_k \quad (5.1)$$
    by employing Algorithm 2.
S7: Set $x_{k+1} = x_k + \alpha_k d_k$, $k = k + 1$ and go to Step S1.
```

In step S1 of the algorithm, a subsample of data is generated to estimate the gradient of the objective function. After a sample is generated, it is retained for the next m iterations. This means that the same subsample is used to estimate the gradient during m iterations, before a new sample $\mathcal{N}_k$ is generated. Each subsample is of size S, which is assumed to be significantly smaller than the size of the full sample - N. As explained in more detail in Chapter 2, this approach of using only a subsample in each iteration can significantly speed up the model training process, especially when working with large datasets, because the gradient is calculated from a smaller number of samples compared to the size of the entire dataset. The estimation of the gradient based on the subsample is performed in step S2. After that, in step S3, the long BB step size - c_k is calculated, in the case when $m > 1$ and the subsample is unchanged, and in the case when $m = 1$ at each iteration. In the case when $m > 1$ and a new subsample was generated in the iteration, the value $\frac{1}{||g_k||}$ is taken for c_k. This is not crucial for the analysis, but from a practical point of view, it may prove useful for the stability and efficiency of the algorithm under real conditions. Also, some other choices are relevant, such as, for example $c_k = 1$. When $m = 1$ the method reduces to a stochastic gradient method with BB choice of the step length similar to one considered in [12]. As already mentioned in the previous chapter, spectral coefficients show non-monotonic behavior during the convergence process, also they can be negative values. For this reason, in step S4, the authors of this algorithm decided to "tame" them by projecting them onto an arbitrary large interval $[\gamma_{min}, \gamma_{max}]$ and dividing them by k. Then, in step S5, the negative gradient is scaled according to the obtained value for γ_k and thus the search direction, d_k, is obtained. At the very end, a non-monotonic procedure for determining the step length is carried out, which is described in detail in Algorithm 3. This procedure, which has proven to be very effective in practice, is used to determine the appropriate step length in each iteration of the algorithm, using estimates of the stochastic function, as in the current iteration as well as in the trial new iteration. Those estimates are obtained by averaging the objective function values over the same set of samples used to estimate the gradient.

Algorithm 3 LSP (Line Search Procedure)

```
**0 Input parameters:** $x_k \in \mathbb{R}^n, \eta \in (0,1), \mathcal{N}_k, \gamma_k, t_k$
**S1 Initialization:** Set $j = 0$, $dm_k = g_k^T d_k$, $\alpha_j = 1$.
**S2:** If
    $$f_{\mathcal{N}_k}(x_k + \alpha_j d_k) \leq f_{\mathcal{N}_k}(x_k) + \eta \alpha_j dm_k + t_k \quad (5.2)$$
go to Step S5. Else, go to Step S3.
**S3:** If $\alpha_j \geq 0.1$, compute
    $$\widetilde{\alpha}_j = \frac{-dm_k \alpha_j^2}{2(f_{\mathcal{N}_k}(x_k + \alpha_j d_k) - f_{\mathcal{N}_k}(x_k) - \alpha_j dm_k)}. \quad (5.3)$$
If $\widetilde{\alpha}_j < 0.1\alpha_j$ or $\widetilde{\alpha}_j > 0.9\alpha_j$,set $\widetilde{\alpha}_j = \alpha_j/2$. Set $\alpha_{j+1} = \widetilde{\alpha}_j$, $j = j + 1$ and go to Step S2.
**S4:** If $\alpha_j < 0.1$, set $\alpha_{j+1} = \alpha_j/2$, $j = j + 1$ and go to Step S2.
**S5:** Set $\alpha_k = \alpha_j$ and STOP.
```

In step S2 of Algorithm 3, the value α_j that satisfies the Armijo condition (5.2) is searched for. If the individual value α_j does not satisfy the Armijo condition, and at the same time is greater than 0.1, the quadratic interpolation $q(x) = ax^2 + bx + c$ is applied. This interpolation is used so that $f_{\mathcal{N}_k}(x)$ and $q(x)$ share two points (x_k and $x_k + \alpha_j d_k$) and have the same derivative at one of them. After that, the optimal value of this quadratic function - $\widetilde{\alpha}_j$ is calculated, which becomes a candidate for

α_j in case the inequalities $\widetilde{\alpha}_j < 0.1\alpha_j$ and $\widetilde{\alpha}_j > 0.9\alpha_j$ are not satisfied. In all remaining cases, Algorithm 3 switches to the backtracking procedure. As can be noticed, one does not have to resample, but the current sample is used to evaluate the estimation function at new points that are used to check if Armijo condition is satisfied.

5.1 Convergence analysis

Now that we are familiar with the SLiSeS algorithm, we will introduce the assumptions under which this method converges to a stationary point. In this thesis, the focus will be on the case when sampling in step S1 of Algorithm 2 is uniform, that is, when each sample has the same probability of being selected. This is important to emphasize because uniform sampling has its specificities and can affect the behavior of the algorithm. First of all, we assume that function f is twice continuously differentiable with Lipschitz-continuous gradients by making assumption on the functions f_i as follows.

Assumption 1. *The functions f_i, $i = 1, \ldots, N$, are bounded from below and twice continuously differentiable with L-Lipschitz-continuous gradients.*

The well-definedness of LSP, as well as the fact that the step length computed by LSP remains bounded away from zero, is confirmed by the following lemma.

Lemma 1. *Suppose that Assumption 1 holds. Then the LSP procedure is well-defined and there exists a constant $\alpha_{min} > 0$ such that $\alpha_{min} \leq \alpha_k \leq 1$ for every k.*

Let us denote by $\mathcal{F}_k$ a σ-algebra generated by $\mathcal{N}_0, ..., \mathcal{N}_{k-1}$. Intuitively, $\mathcal{F}_k$ includes all possible events that can be inferred or derived based on the information from samples $\mathcal{N}_0, ..., \mathcal{N}_{k-1}$. We can note that x_k is $\mathcal{F}_k$-measurable, that is, we have complete information about the value of x_k provided we know $\mathcal{F}_k$. Additionally, it should be noted that α_k, g_k and γ_k do not belong to the set of $\mathcal{F}_k$-measurable variables. Our focus will be exclusively on $\mathcal{F}_k$ during iterations k where k satisfies the condition $\mod(k, m) = 0$. We will refer to these iterations as outer iterations. In the remaining iterations, to be referred to as inner iterations, we keep the same sample. Based on the uniform sampling assumption, it follows that:

$$E(f_{\mathcal{N}_k}(x_k)|\mathcal{F}_k) = f(x_k) \tag{5.4}$$

and

$$E(g_k|\mathcal{F}_k) = \nabla f(x_k). \tag{5.5}$$

Next, we state another assumption that is common to subsampling methods.

Assumption 2. *There exists a constant $G > 0$ such that for all k there holds:*

$$E(||g_k||^2|\mathcal{F}_k) \leq 2(G + ||\nabla f(x_k)||^2).$$

Given that $\mathcal{N}_k$ is finite let s denote the number of possible samples $\mathcal{N}_k$: $\mathcal{N}_k^1, ..., \mathcal{N}_k^s$. Then, for any $\mathcal{N}_k$ we have $||\nabla f(x_k)||^2 = E[\nabla f(x_k)^T \nabla f_{\mathcal{N}_k}(x_k)|\mathcal{F}_k]$ and

$$||\nabla f(x_k)||^2 = \frac{1}{s}\left(\sum_{i\in I_k} \nabla f(x_k)^T \nabla f_{\mathcal{N}_k^i}(x_k) + \sum_{i\in J_k} \nabla f(x_k)^T \nabla f_{\mathcal{N}_k^j}(x_k)\right), \quad (5.6)$$

where $I_k = \{i : \nabla f(x_k)^T \nabla f_{\mathcal{N}_k^i}(x_k) \geq 0\}$ and $J_k = \{j : \nabla f(x_k)^T \nabla f_{\mathcal{N}_k^j}(x_k) < 0\}$. It can be noticed from (5.6) that $I_k \neq \emptyset$ for all k.

Finally, to ensure existence of a.s. (almost sure) convergent subsequence to the stationary point, the following assumption is also needed.

Assumption 3. *There exist constants $C, \theta > 0$ and $0 < \delta < 1$ such that*

$$P(B_k)E[\nabla f(x_k)^T \nabla f_{\mathcal{N}_k}(x_k)|\mathcal{F}_k, B_k] \leq \frac{\theta}{s}||\nabla f(x_k)||^2 + \frac{C}{sk^\delta},$$

where B_k is event that $\nabla f(x_k)^T \nabla f_{\mathcal{N}_k}(x_k) \geq 0$ and $P(B_k)$ is the probability of this event.

If assumptions 1-3 hold, we have the following theorem.

Theorem 7. *Suppose that the Assumptions 1-3 hold and let $\{x_k\}$ be a sequence generated by the SLiSeS algorithm. Then, provided that $s \geq \theta$, with probability 1 there holds*

$$\liminf_{k\to\infty} \|\nabla f(x_k)\| = 0.$$

Theorem 7 shows that for the SLiSeS algorithm we have that $\liminf_{k\to\infty} \|\nabla f(x_k)\| = 0$. This means that SLiSeS algorithm generates a subsequence such that it converges to a stationary point of the function f. It is possible to get stronger result on convergence of the SLiSeS algorithm to the solution of the problem (3.1), by adding following assumption.

Assumption 4. *The function f is strongly convex.*

Theorem 8. *Suppose that the assumptions of Theorem 7 hold together with assumption A4. Then, the sequence $\{x_{mk-1}\}_{k\in\mathbb{N}}$ converges to the solution of problem* (3.1) *x^* almost surely.*

Theorem 8 states that sequence of outer iterations $\{x_{mk-1}\}_{k\in\mathbb{N}}$ generated by SLiSeS algorithm converges to the solution with probability 1.

6 Numerical results

In this chapter, the numerical results obtained by application of the SLiSeS algorithm in several different scenarios will be presented. The results will be displayed in the form of graphics, which will give us a more detailed insight into the performance of the algorithm in different scenarios.

In the experiments, the focus will be on strongly convex quadratic functions and on L2-regularized logistic regression problems, as canonical problems that satisfy the assumptions stated in the previous chapter.

The considered quadratic functions, for $1 \leq i \leq N$, are given by

$$f_i(x) = \frac{1}{2}(x - b_i)^T A_i (x - b_i), \tag{6.1}$$

where $b_i \in \mathbb{R}^n$, and $A_i \in \mathbb{R}^{n,n}$ is a symmetric positive definite matrix. Matrices A_i and the vectors b_i are obtained as in [18], i.e., vectors b_i are extracted from the Uniform distribution on $[1, 31]$, independently from each other. Matrices A_i are of the form $A_i = Q_i D_i Q_i^T$, where D_i is a diagonal matrix with Uniform distribution on $[1, 101]$ and Q_i is the matrix of orthonormal eigenvectors of $\frac{1}{2}(C_i + C_i^T)$, and the matrix C_i has components drawn independently from the standard Normal distribution. This construction guarantees that the matrices A_i are symmetric positive definite, and this whole process ensures the generation of different and independent sets of matrices and vectors, which are necessary for experimentation and analysis.

For the logistic regression, for $1 \leq i \leq N$, the function f_i is given by

$$f_i(x) = \log(1 + \exp(-b_i(a_i^T x))) + \frac{\lambda}{2}||x||_2^2, \tag{6.2}$$

where the vectors $a_i \in \mathbb{R}^n$, and the labels $b_i \in \{-1, 1\}$ are given, and the regularization parameter λ is set to 10^{-4}. The vectors a_i, and the labels b_i are obtained from three real datasets: Adult [5], Cina0 [6] and Voice dataset[22].

Dataset	n	N
ADULT	123	32561
CINA0	132	16033
VOICE	309	126

Table 1: Properties of the datasets used in the experiments.

The Adult dataset exclusively comprises binary features, offering insights into socio-economic factors and their association with income levels. The target variable in this dataset indicates whether an individual earns more than $\$50,000$.

On the other hand, the Cina0 dataset provides a more extensive array of information compared to the Adult dataset, including both binary and numerical features. While binary features provide insights into particular socio-economic factors, numerical characteristics enable a more in-depth examination of quantitative variables like years of education or weekly working hours.

The Voice dataset evaluates whether voice rehabilitation treatments lead to phonations classified as "acceptable"or "unacceptable,"presenting a binary classification task. This dataset consists entirely of numerical features.

We scaled the previously unscaled numerical data by normalizing each attribute vector. This ensured their consistency and relevance in the analysis. Scaling the numerical features ensures that each feature contributes equally to the analysis and prevents features with larger scales from dominating the learning process.

In our experiments, we report the computational cost, which is measured by the number of function evaluations relative to the observed reduction of the objective function, as in [12]. First of all, it can be noticed that the main cost in the computation of each function f_i is the evaluation of $\exp(-b_i(a_i^T x))$. Computing the gradient of f_i for an arbitrary index i, we obtain

$$\nabla f_i(x) = \frac{\exp(-b_i(a_i^T x))}{1 + \exp(-b_i(a_i^T x))} - b_i a_i + \lambda x, \tag{6.3}$$

thus the evaluation of the gradient comes for free from the evaluation of the corresponding function. Based on this observation, at the beginning of each iteration,the values of $\exp(-b_i(a_i^T x))$ are computed, and subsequently utilized throughout the iteration to assess both the sampled function $f_{\mathcal{N}_k}(x_k)$ and the sampled gradient $\nabla f_{\mathcal{N}_k}(x_k)$.

In the case when the functions f_i are strongly convex quadratic functions, as defined in (6.1), we obtain $\nabla f_i = A_i(x - b_i)$. Following a similar procedure as in the previous case, at the beginning of each iteration, the values of $A_i(x - b_i)$ are first computed, and then used within the iteration to obtain the values of $f_{\mathcal{N}_k}(x_k)$, and $\nabla f_{\mathcal{N}_k}(x_k)$.

Therefore, to compute the cumulative number of function evaluations, in our experiments, the counter is incremented by $S = |\mathcal{N}_k|$ each time the function $f_{\mathcal{N}_k}$ is evaluated. Since evaluating the gradient does not incur any additional computational costs, it is not separately counted in the total number of evaluations.

All runs are terminated when a maximum number of iterations ($maxiter$) is reached. The certain input parameters in the SLiSeS algorithm are fixed as follows: $x_0 = 0$, $\eta = 10^{-4}$ and $t_k = 1/2^k$. At the beginning of each subsection, it will be stated how the remaining parameters, which are not the subject of investigation in current subsection, are fixed.

6.1 Changing the number of internal iterations m

Firstly, we investigate the behavior of the SLiSeS algorithm for different values of the number of internal iterations m, in which we keep the same sample. For all experiments, we set $S = |\mathcal{N}_k| = 1$, $\gamma_{min} = 10^{-8}$, $\gamma_{max} = 10^8$, and the maximum number of iterations - $maxiter = 50$.

6.1 Changing the number of internal iterations m

In the paper [11], it has been shown that the SLiSeS algorithm exhibits better behavior in cases when $m > 1$ compared to $m = 1$. Therefore, our objective is to explore scenarios where $m > 1$ and make corresponding comparisons.

Let us start by considering strictly convex quadratic functions. Figures 6 and 7 present the performance of the SLiSeS algorithm for several values of m, and for two distinct values of n: 10 and 100, respectively. It can be noticed that in the both cases, for $m = 3$, $m = 4$, $m = 5$, performances are better than for $m = 2$. In the case when $n = 10$, algorithm performances for $m = 10$ are slightly worse than for the remaining choices of m. However, when $n = 100$, algorithm performances for $m = 10$ are similar to those for $m = 3$, $m = 4$, $m = 5$.

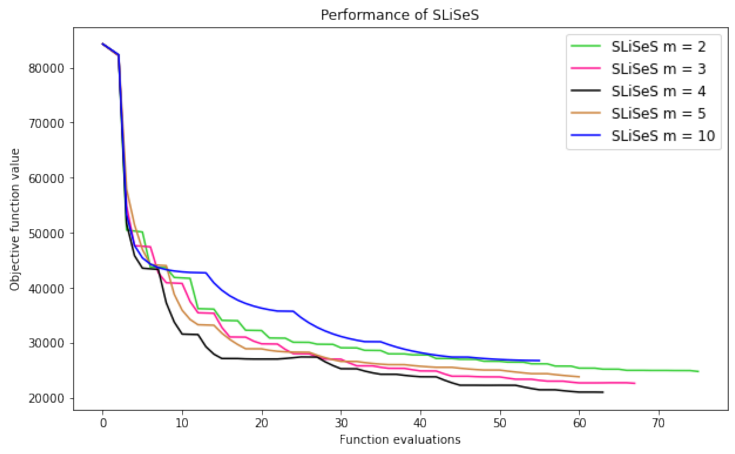

Figure 6: Performance of SLiSeS on strictly convex quadratics for $m = 2,\ 3,\ 5,\ 10,$ $N = 1000$, $maxiter = 50$, and $n = 10$.

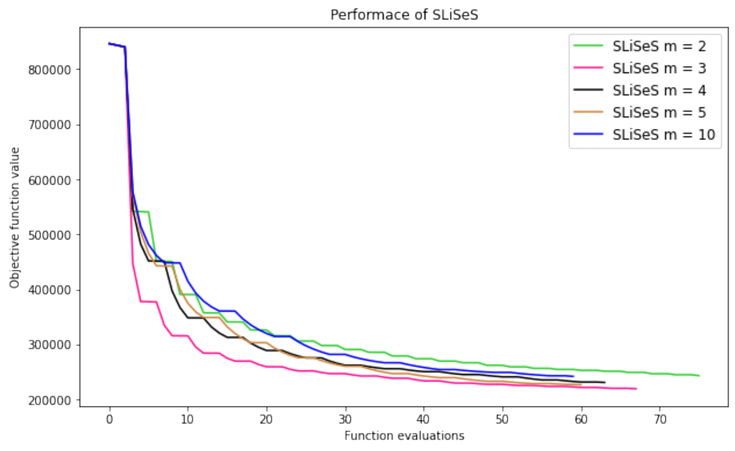

Figure 7: Performance of SLiSeS on strictly convex quadratics for $m = 2, 3, 5, 10,$ $N = 1000$ $maxiter = 50$, and $n = 100$.

6.1 Changing the number of internal iterations m

In Figures 8, 9, and 10, we focus our attention on observing how the algorithm behaves when changing the values of m in the case of logistic regression problems (Adult, Cina0, and Voice datasets). It can be noticed that the implemented SLiSeS algorithm performs different on the different datasets. For instance, we notice that the algorithm achieves the best performance on the Adult and Cina0 datasets when $m = 10$, while on the Voice dataset, $m = 10$ exhibits worse algorithm performance compared to the cases when $m = 2$, $m = 3$, $m = 4$, $m = 5$.

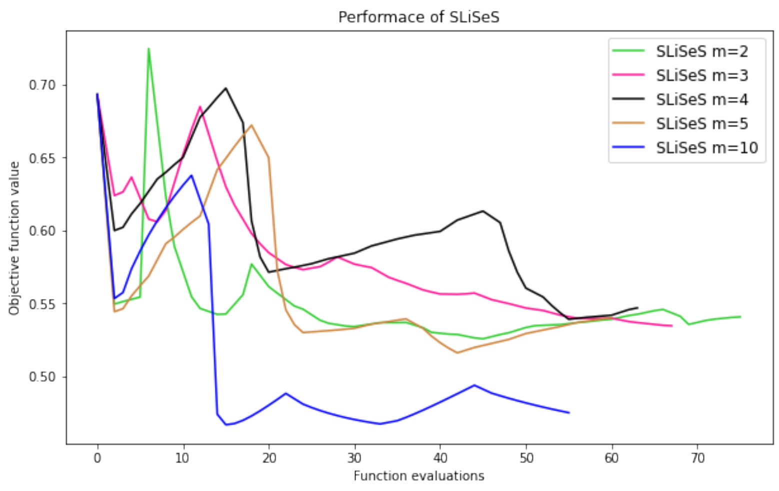

Figure 8: Performance of SLiSeS on the on the Adult dataset for $m = 2, 3, 4, 5, 10$, and $maxiter = 50$.

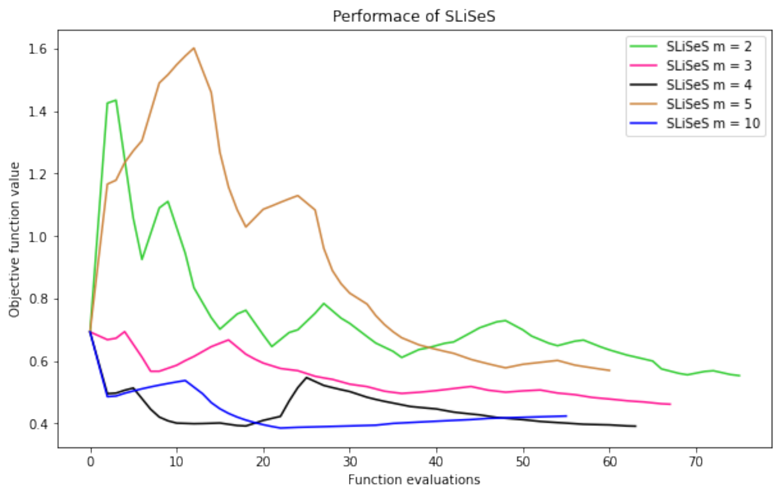

Figure 9: Performance of SLiSeS on the on the Cina0 dataset for $m = 2, 3, 4, 5, 10$, and $maxiter = 50$.

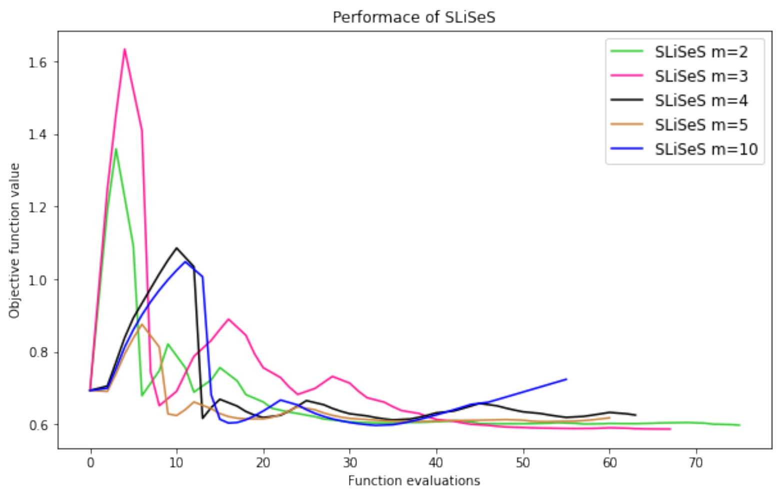

Figure 10: Performance of SLiSeS on the on the Voice dataset for $m = 2, 3, 4, 5, 10$, and $maxiter = 50$.

From the conducted experiments we can conclude that there is no choice of m value for which the algorithm's performance will be optimal for all of the above cases. However, it can be noticed that on average, it appears that the value $m = 3$ shows better performance compared to other choices for the value of m.

6.2 Changing the size of subsample S

In this subsection, we investigate how the algorithm behaves depending on the size of the subsamples ($S = |\mathcal{N}_k|$) it uses. We report the computational cost measured by the number of function evaluations, and by the number of iterations, versus the observed decrease in the objective function. We conducted tests on strictly convex quadratics when $n = 10$, as well as on linear regression problems (Cina0 and Voice datasets). In all experiments, the values are fixed as follows: $m = 3$, $\gamma_{min} = 10^{-8}$, $\gamma_{max} = 10^{8}$.

When measuring computational cost by the number of function evaluations, in order to compare the algorithm's performances over the same interval, different values for maxiter are set for different values of S. When the parameter S is small, a large number of iterations is required, while increasing S reduces the number of iterations. On the other hand, when computational cost is measured by the number of iterations, maxiter is set to 50.

Figures 11, 12 show the performance of the SLiSeS algorithm on strictly convex quadratics.

6.2 Changing the size of subsample S

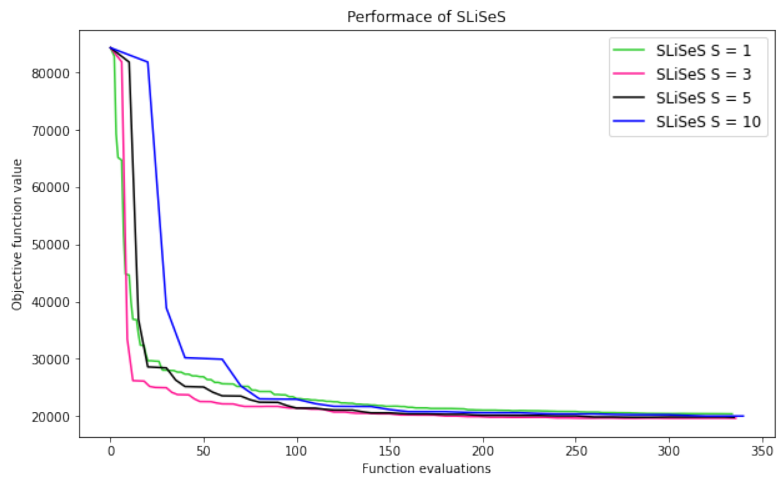

Figure 11: Performance of SLiSeS on strictly convex quadratics for S = 1, 3, 5, 10, N = 1000, maxiter = 250, 85, 50, 25 and n = 10.

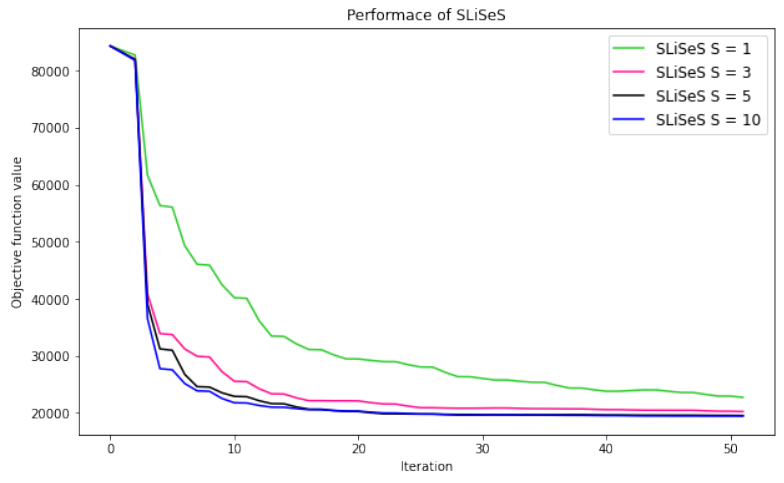

Figure 12: Performance of SLiSeS on strictly convex quadratics for S = 1, 3, 5, 10, N = 1000, maxiter = 50 and n = 10.

Figure 11 shows that after a certain number of function evaluations, the algorithm's performances are almost equal for all values of S. On the other hand, when considering the number of iterations (Figure 12), as it is expected, we can see that the SLiSeS performs better as S is higher, since more information about function is available. However, it can been seen that there is no much difference between $S = 3$, $S = 5$ and $S = 10$.

6.2 Changing the size of subsample S

Figures 13 and 14 show performances of SLiSeS on the Cina0 dataset, while Figures 15, and 16 show performances of SLiSeS on the Voice dataset.

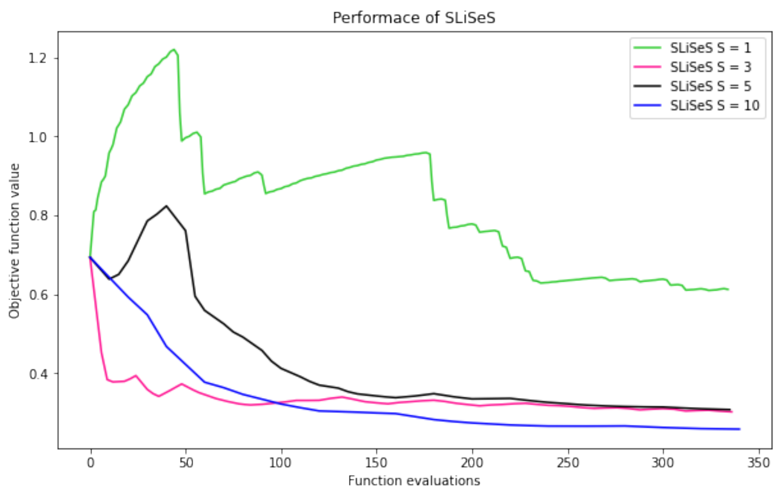

Figure 13: Performance of SLiSeS on the Cina0 dataset for S = 1, 3, 5, 10, and maxiter = 250, 85, 50, 25.

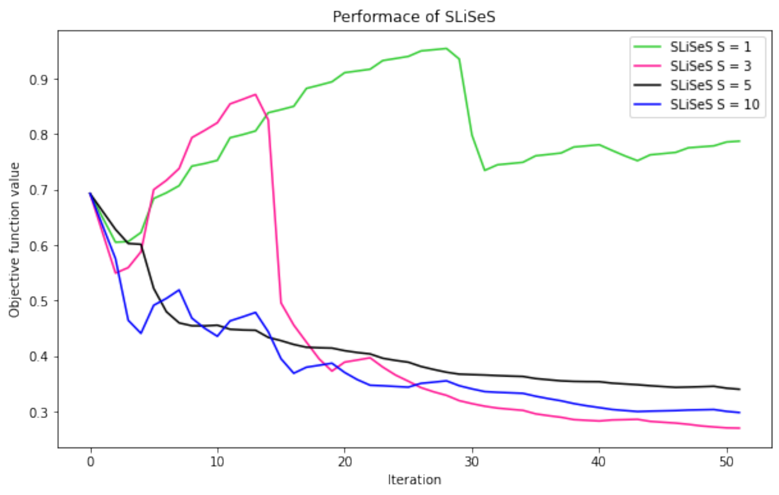

Figure 14: Performance of SLiSeS on the on the Cina0 dataset for S = 1, 3, 5, 10, and maxiter = 50.

When examining the logistic regression problem using the Cina dataset, Figures 13 and 14 clearly show that the performance of the algorithm decreases significantly for $S = 1$, compared to $S = 3, 5, 10$.

6.2 Changing the size of subsample S

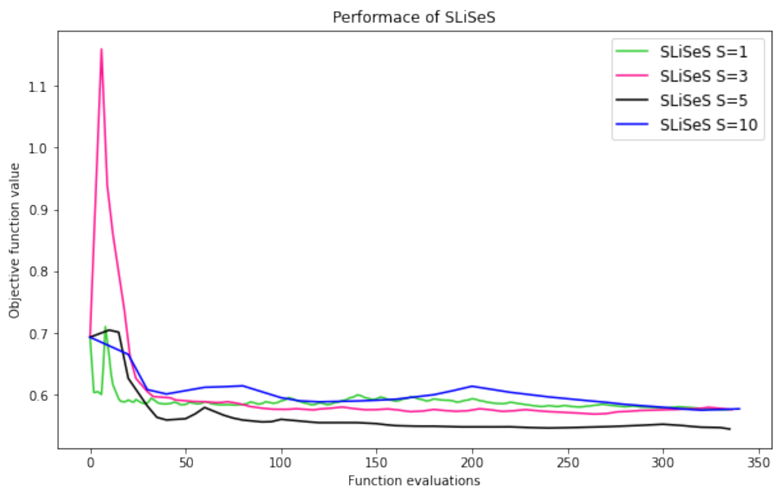

Figure 15: Performance of SLiSeS on the on the Voice dataset for S = 1, 3, 5, 10, and maxiter = 250, 85, 50, 25.

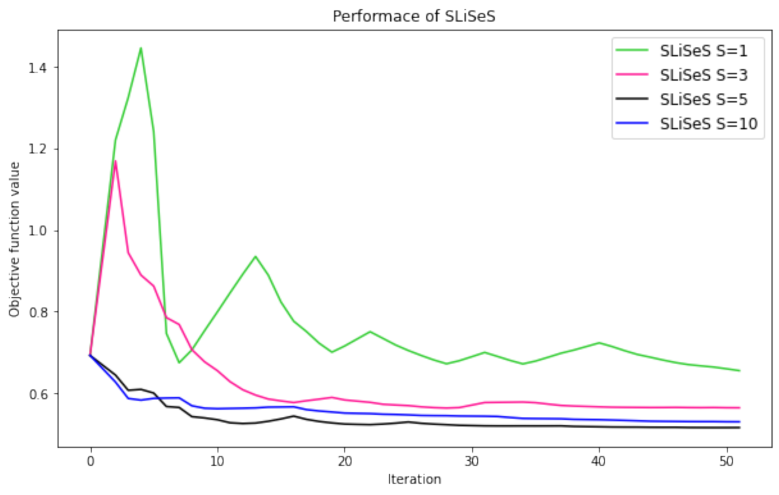

Figure 16: Performance of SLiSeS on the on the Voice dataset for S = 1, 3, 5, 10, and maxiter = 50.

From the specific numerical experiments, it can be noticed that when measuring computational cost based on the number of iterations, the performances of the SLiSeS algorithm for $S = 1$ exhibit the poorest performance (Figures 12, 14, and 16). Moreover, when comparing the performance of the algorithm for $S = 3$, $S = 5$ and $S = 10$, it can been seen that there is no an optimal choice of S, since for different datasets, the best performance of algorithm is obtained for different choice of S. However, due to the computational cost, it is better to use the values $S = 3$ and $S = 5$, since they offer sufficiently good results with faster execution compared to the higher values such as $S = 10$.

6.3 Thresholds γ_{min} and γ_{max}

As previously stated, the spectral coefficients - c_k may exhibit both the negative and excessively large values. Therefore, it becomes necessary to impose constraints in the form of lower (γ_{min}) and upper (γ_{max}) bounds, such that $0 < \gamma_{min} \leq 1 \leq \gamma_{max} < \infty$. More precisely, within each iteration of the SLiSeS algorithm, the spectral coefficient c_k is calculated (Step S3), and then, in the next step, γ_k is chosen such that it holds $\gamma_k = \min\{\gamma_{max}, \max\{\gamma_{min}, c_k\}\}$. For the descending direction, $d^k = \frac{\gamma_k}{k} \nabla f_{\mathcal{N}_k}(x^k)$ is used. In this subsection, we explore how the choice of γ_{min} and γ_{max} influences the algorithm's performance.Throughout all experiments, the parameters $m = 3$, $S = 1$, and $maxiter = 50$ are considered.

The research begins with an analysis of the SLiSeS algorithm's performance on the strictly convex quadratics for $n = 10$ and $n = 100$ (Figures 19 and 20). To assess the impact of thresholds on performance, we initially analyzed the behavior of the c_k value during iterations, as depicted in Figures 17 and 18. It can be noticed the non-monotonic behavior of the spectral coefficients. In addition, it is noted that for $1 \leq k \leq 50$, the condition $10^{-8} < c_k < 1$ holds. Accordingly, we decided to explore the following cases:

1. $\gamma_{min} = 10^{-8}$ and $\gamma_{max} = 10^{8}$ - In this case, the spectral coefficient is used in each iteration, because $\gamma_{min} < c_k < \gamma_{max}$. Therefore, the value of c_k is directly used to determine γ_k.

2. $\gamma_{min} = 10^{-2}$ and $\gamma_{max} = 10^{2}$ - Here, if $c_k < \gamma_{min}$, c_k is used; otherwise γ_{min} is used. This approach enables us to observe how the algorithm behaves when spectral coefficients are not utilized in every iteration, given the constraints.

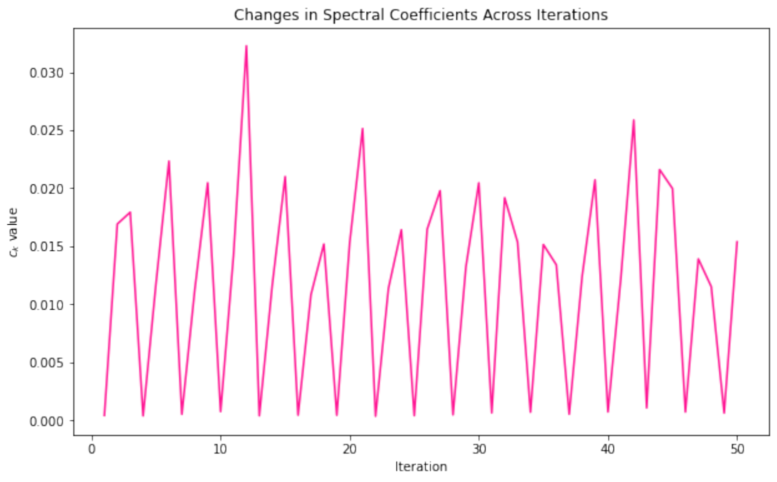

Figure 17: c_k across iterations for $n = 10$ and $N = 1000$.

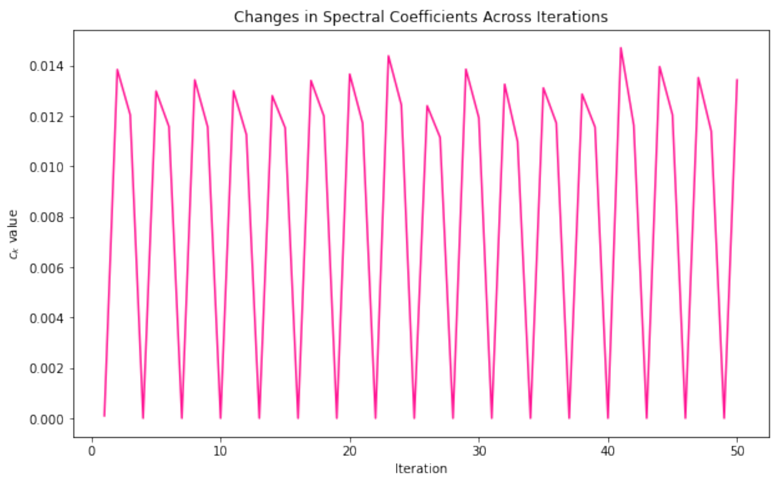

Figure 18: c_k across iterations for $n = 100$ and $N = 1000$.

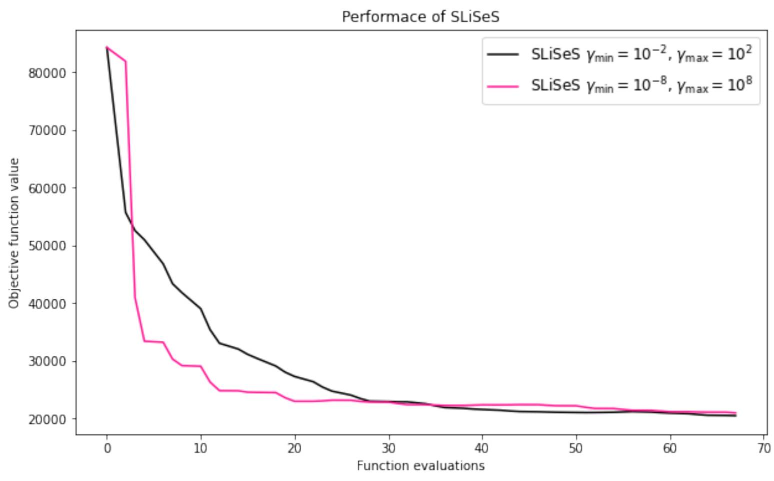

Figure 19: Performance of SLiSeS on strictly convex quadratics for N = 1000, maxiter = 50 and n = 10.

Figures 19 and 20 show that the SLiSeS algorithm exhibits better convergence behavior when thresholds are used, compared to the case when c_k is used in each iteration.

6.3 Thresholds γ_{min} and γ_{max}

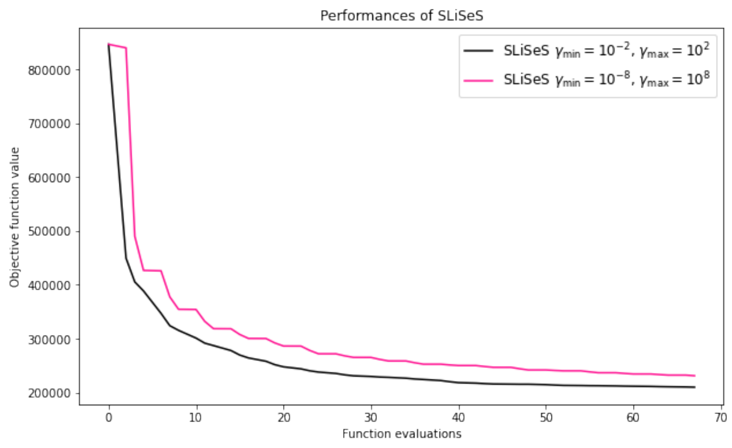

Figure 20: Performance of SLiSeS on strictly convex quadratics for N = 1000, maxiter = 50 and n = 100.

Figures 21 and 22 show c_k values across the iterations while Figures 23 and 24 show the performance of the algorithm for the different thresholds, for Cina0 and Voice datasets, respectively. As it can be seen, using the same reasoning as in the quadratic case, existence of thresholds again has positive impact on the performance of algorithm for the both datasets.

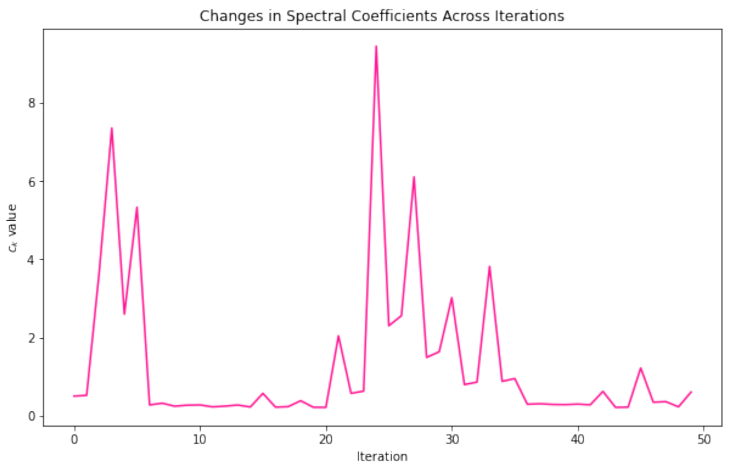

Figure 21: c_k values across iterations for Cina0 dataset.

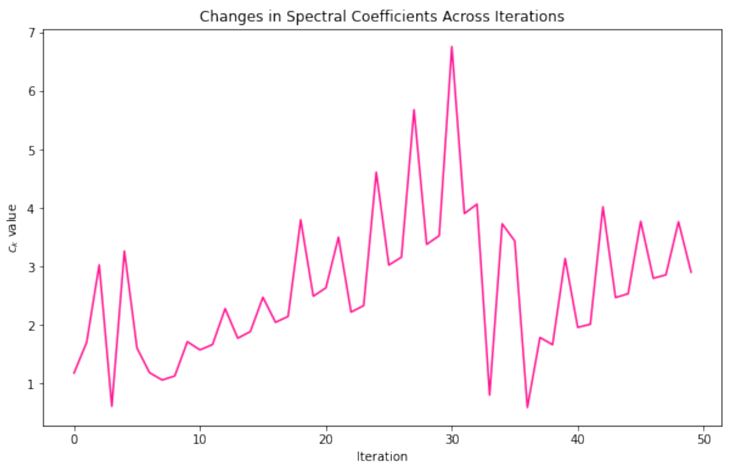

Figure 22: c_k values across iterations for Voice dataset.

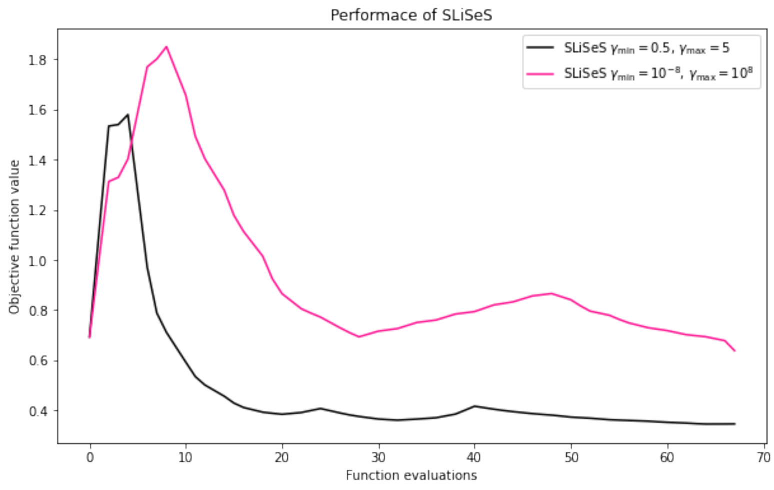

Figure 23: Performance of SLiSeS on the on the Cina0 dataset for maxiter = 50.

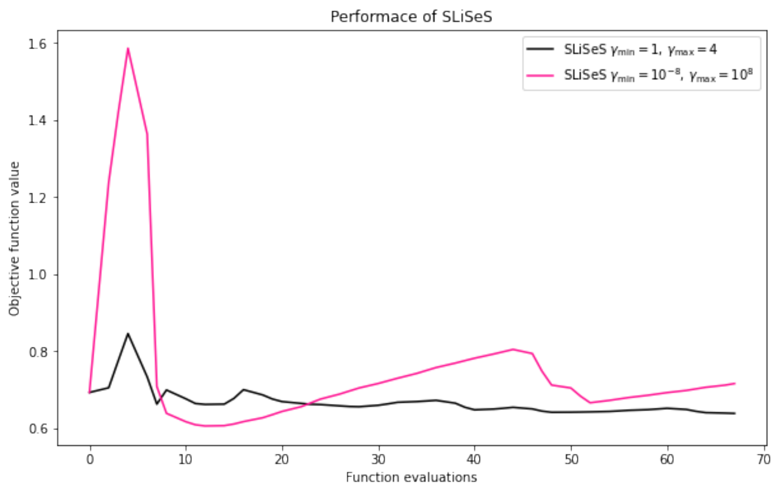

Figure 24: Performance of SLiSeS on the on the Voice dataset for maxiter = 50.

6.4 Comparison of SGD and SLiSeS

In this subsection, the SLiSeS method will be compared with its competitor for solving problem 3.1 - the Stohastics Gradient Descent (SGD). The SGD can be interpreted as a simplified case of the SLiSeS method where the step length γ_k is obtained either as a sufficiently small constant to guarantee convergence, or in a diminishing way such that $\sum \gamma_k = \infty$, and $\sum \gamma_k^2 < \infty$ to guarantee convergence (the most common choice is $\frac{1}{k}$). In either case, the SGD method does not require a line search strategy. The Figures 25, 26, and 27 show the performance of these two methods applied to a strictly convex quadratic when $n = 100$ and $N = 1000$, as well as on the Adult and Voice datasets, respectively. Throughout all experiments, value of S is set to 3.

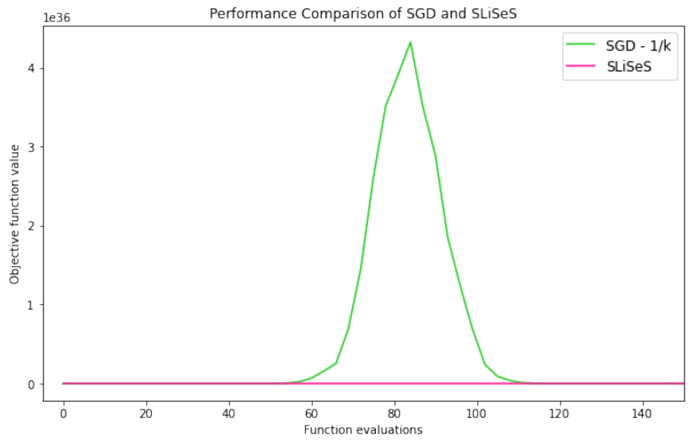

Figure 25: Performance of the SLiSeS method $(m = 3)$ and the SGD method with step length $1/k$, for $maxiter = 50$, on a strictly convex quadratic when $n = 100$ and $N = 1000$.

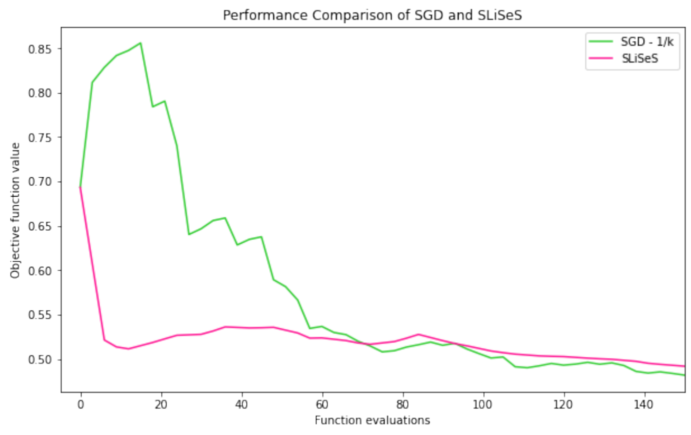

Figure 26: Performance of the SLiSeS method $(m = 3)$ and the SGD method with step length $1/k$, for $maxiter = 50$, on the Adult dataset.

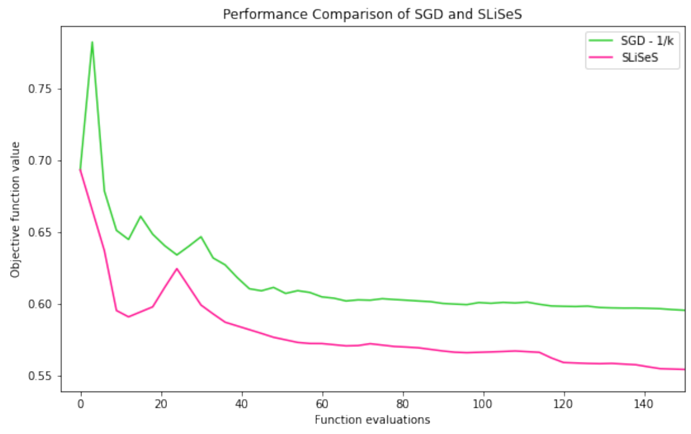

Figure 27: Performance of the SLiSeS method $(m = 3)$ and the SGD method with step length $1/k$, for $maxiter = 50$, on the Voice dataset.

When comparing the results obtained from applying the SGD and SLiSeS algorithms, it is evident that SLiSeS exhibits better convergence behavior towards the stationary point compared to SGD. This is because the SLiSeS algorithm is less affected by noise and more adaptable to different types of the data.

7 Conclusions

Solving large-scale finite-sum optimization problems is a challenging but key area in machine learning. With the advent of massive amounts of data, traditional optimization methods have become impractical or inefficient due to high computational requirements and memory issues. Therefore, stochastic algorithms become the most suitable options.

Among the stochastic algorithms based on negative gradient direction is the SGD and its variants, which, although possessing many theoretical advantages, in practice can face several challenges that may affect its efficiency and performance. By focusing on practical behavior, these methods have been upgraded to overcome these challenges. Thus, gradient-type stochastic methods were developed that included the use of non-monotonic step lengths such as the Barzilai and Borwein (BB) spectral ones. This method has proven to be extremely effective in practice and is the subject of numerous scholarly works in order to gain a deeper understanding of its behavior and application in different optimization contexts.

However, what all these stochastic gradient methods have in common is that the sample is changed at each iteration. Consequently, the authors of the Subsampled Line Search Spectral Gradient Method for Finite Sums proposed keeping the same subsample during several iterations, which makes it possible to use more information from the same data sample and the possibility of reducing the computational cost. After conducting numerical experiments, it was noticed that the spectral method with this method of subsampling improves the existing gradient methods.

In this research, we analyzed the behavior of the SLiSeS algorithm with different parameters. The goal was to study how parameter variations affect the performance of this algorithm in solving certain optimization problems. Based on our experiments, we conclude that it is important to carefully select algorithm parameters according to the specific requirements of the problem and data characteristics. Further research and analysis are needed to better understand these issues and enable more efficient use of the SLiSeS algorithm in practice. These findings provide useful guidelines for optimizing algorithm performance in real-world machine learning applications.

The Python code

Code for Adult, Cina0 and Voice datasets.

```
def sum_(data, x, lamb):
    fsum=0
    sum_grad=0

    for i in range(data.shape[0]):
        a= data.iloc[i][:-1].to_numpy()              #vectors
        b= data.iloc[i][data.iloc[i].shape[0]-1]    #labels

        exp = np.exp((-1 * (b * np.dot(a, x))))
        loga = np.log(1 + exp)
        reg = 0.5 * lamb * np.linalg.norm(x)**2

        fsum += (loga + reg) / data.shape[0]

        sum_grad += ((exp / (1 + exp)) * -b * a +  lamb * x) / data
    .shape[0]
    return fsum, sum_grad

def algorithm_SLiSeS_regression(data, x, eta, S, m,  gamma_min,
    gamma_max, lamb, maxiter):

    k=1
    evaluations=0
    performance_data=[] #f(x_k)
    evaluations_list=[0]
    alpha_seq=[]
    ck_list=[]
    gamma_list=[]
    num_of_iter=[0]
    N=list(range(0, data.shape[0]))

    while k<=maxiter:

        f_of_x= sum_(data, x, lamb)[0]
        performance_data.append(f_of_x)

        tk= 1/2**k

        #STEP 1 - Sampling

        if (k-1) % m == 0:

            subsetN=random.sample(list(N), S)
        else:

            subsetN = subsetN

        #STEP 2 - Computing gradient

        new_data = data.iloc[subsetN, :]
        if (k-1) % m == 0:

            f_Nk, gk = sum_(new_data, x, lamb)
            evaluations+=S

        else:
            f_Nk= new_f_Nk_x
            gk=new_gk

        #STEP 3 - Spectral coefficient

        if  (k-1)%m ==0 and m>1:

            ck = 1/np.linalg.norm(gk)

        else:

            s_km1 = x - x_prev
            y_km1 = gk - gk_prev
            ck = np.linalg.norm(s_km1)** 2 / np.dot(s_km1, y_km1)

        #STEP 4

        gamma_k= (min(gamma_max, max(gamma_min, ck)))/k


        #STEP 5 - Search direction

        dk = -(gamma_k * gk)

        #STEP 6 - Step size

        alpha_k, evaluations_lsp, new_f_Nk_x, new_gk =
    algorithm_LSP_regression(x, eta,  gamma_k, tk, dk, gk, new_data
    , f_Nk, S,lamb)
        evaluations+= evaluations_lsp
        alpha_seq.append(alpha_k)

        #STEP 7

        x_prev = x.copy()
        gk_prev = gk.copy()
        x = x + alpha_k * dk
        k+=1
        num_of_iter.append(k)
        evaluations_list.append(evaluations)
        if k == (maxiter +1):

            performance_data.append(sum_(data, x, lamb)[0])

        ck_list.append(ck)
        gamma_list.append(gamma_k)

    return  performance_data, evaluations_list, alpha_niz, ck_list,
     gamma_list, num_of_iter

def algorithm_SGD(data, x, eta, S, lamb, maxiter):

    k=1
    evaluations=0
    performance_data=[]
    evaluations_list=[0]
    num_of_iter=[0]
    N=list(range(0, data.shape[0]))

    while k<=maxiter:

        f_of_x= sum_(data, x, lamb)[0]
        performance_data.append(f_of_x)
        #STEP 1 - Sampling

        subsetN=random.sample(list(N), S)

        #STEP 2 - Computing gradient

        new_data = data.iloc[subsetN, :]
        f_Nk, gk = sum_(new_data, x, lamb)
        evaluations+=S

        #STEP 3 - gamma
        gamma_k= 1/k

        #STEP 4 - Search direction

        dk = -(gamma_k * gk)

        #STEP 5

        x_prev = x.copy()
        gk_prev = gk.copy()
        x= x + dk
        k+=1
        num_of_iter.append(k)
        evaluations_list.append(evaluations)
        if k == (maxiter +1):

            performance_data.append(sum_(data, x, lamb)[0])

    return  performance_data, evaluations_list,  num_of_iter

def algorithm_LSP_regression(xk, eta,  gamma_k,tk, dk, gk, new_data
    , f_Nk, S, lamb):

    evaluations2=0
    dmk= np.dot(dk, gk)
    alphaj=1

    while True:

        f_Nk_a , g_Nk_a= sum_(new_data, xk + (alphaj * dk), lamb)
        evaluations2+=S

        if f_Nk_a <= f_Nk +eta*alphaj*dmk+ tk and alphaj <=1:

            alphak= alphaj
            return alphak, evaluations2, f_Nk_a , g_Nk_a

        else:

            if alphaj>0.1:

                alphaj_new=(-dmk * (alphaj)**2) / (2*(f_Nk_a - f_Nk
    - alphaj*dmk))

                if alphaj_new<0.1*alphaj or alphaj_new>0.9*alphaj:

                    alphaj_new=alphaj/2

                alphaj= alphaj_new

            else:
                alphaj=alphaj/2
```

Code for sum of strictly convex quadratic functions.

```
def matrix_A(n): #n is dimension
    A = np.zeros((n,n))
    D = np.diag(np.random.uniform(1, 101, size=(n,)))
    C = np.random.normal(size=(n, n))
    E = 0.5 * (C + C.T)
    Q = eigenvectors
    A = np.dot(Q, np.dot(D, Q.T))
    return A

def vector_b(n):
    b = np.random.uniform(1, 31, size=(n,)) #real
    return b

def quadratic_function1(x):
    s = x.size
    A = matrix_A(s)
    b = vector_b(s)
    return  A, b

def generating_q_functions(N, dimension):
    f=[]
    for i in range(N):
        x=np.random.uniform(size=(dimension,))
        a=quadratic_function1(x)
        A= a[0]
        b=a[1]
        f.append((A,b))
    return f

def sumf(x, t):
    s = 0 #fucntion value
    s2=0  #gradient
    for i in range(len(t)):
        A, b = t[i][0], t[i][1]
        gradient = np.dot(A, x - b)
        s2+= gradient / len(t)
        s += 0.5 * np.dot((x - b).T, gradient) /len(t)
    return s, s2

def algorithm_SLiSeS(parametri_funkcije, x, eta, S, m,  gamma_min,
    gamma_max, maxiter):

    k=1
    S=S
    N=list(range(1, len(par_of_q_f)))
    evaluations=0
    performance_data=[]
    evaluations_list=[0]
    alpha_seq=[]
    num_of_iter=[0]
    ck_list=[]


    while k <= maxiter:

        performance_data.append(sumf(x, parametri_funkcije)[0])
        tk= 1/2**k

        #STEP 1 - Sampling

        if (k-1) % m == 0:

            subsetN=random.sample(list(N), S)
        else:

            subsetN = subsetN

        #STEP 2 - Computing gradient

        sf = [parametri_funkcije[i] for i in subsetN]
        new_func = partial(sumf, t=sf)

        if (k-1) % m == 0:

            f_Nk, gk = new_func(x)
            evaluations+=S

        else:

            f_Nk=new_f_Nk_x
            gk=new_gk

        #STEP 3 - Spectral coefficient

        if  (k-1)%m==0 and m>1:

            ck = 1/np.linalg.norm(gk)

        else:

            s_km1 = x - x_prev
            y_km1 = gk - gk_prev
            ck = np.linalg.norm(s_km1) ** 2 / np.dot(s_km1, y_km1)

        #STEP 4

        gamma_k= (min(gamma_max, max(gamma_min, ck)))/k

        #STEP 5 - Search direction

        dk = -(gamma_k * gk)

        #STEP 6 - Step size

        alpha_k, e_lsp, new_f_Nk_x, new_gk = algorithm_LSP(x, eta,
    gamma_k,tk, dk, gk, new_func, f_Nk, S)
        evaluations+= e_lsp
        alpha_seq.append(alpha_k)

        #STEP 7

        x_prev = x.copy()
        gk_prev = gk.copy()
        x = x + alpha_k * dk
        k+=1
        if k== (maxiter + 1):
            performance_data.append(sumf(x, parametri_funkcije)[0])

        evaluations_list.append(evaluations)
        num_of_iter.append(k)
        ck_list.append(ck)

        return  performance_data, evaluations_list, alpha_seq ,
    num_of_iter, ck_list

def algorithm_SGDQ(parametri_funkcije, x, eta, S, maxiter):
    k=1
    N=list(range(1, len(par_of_q_f)))
    evaluations=0
    performance_data=[]
    evaluations_list=[0]
    num_of_iter=[0]

    while k <= maxiter: #maxiter

        performance_data.append(sumf(x, parametri_funkcije)[0])
        #STEP 1 - Sampling

        subsetN=random.sample(list(N), S)

        #STEP 2 - Computing gradient

        sf = [parametri_funkcije[i] for i in subsetN]
        new_func = partial(sumf, t=sf)
        f_Nk, gk = new_func(x)
        evaluations+=S


        #STEP 3

        gamma_k= 1/k

        #STEP 4 - Search direction

        dk = -(gamma_k * gk)

        #STEP 5

        x_prev = x.copy()
        gk_prev = gk.copy()
        x = x +  dk
        k+=1
        if k== (maxiter + 1):

            performance_data.append(sumf(x, parametri_funkcije)[0])

        evaluations_list.append(evaluations)
        num_of_iter.append(k)

        return  performance_data, evaluations_list,  num_of_iter

def algorithm_LSP(xk, eta,  gamma_k,tk, dk, gk, new_func, f_Nk, S):

    evaluations2=0
    dmk= np.dot(dk, gk)
    alphaj=1

    while True:
        f_Nk_a, g_Nk_a= new_func(xk+alphaj*dk)
        evaluations2+=S
        if f_Nk_a <= f_Nk +eta*alphaj*dmk+tk:

            alphak= alphaj
            return alphak, evaluations2, f_Nk_a , g_Nk_a

        else:
            if alphaj>0.1:

                alphaj_new=(-dmk*(alphaj)**2)/(2*(f_Nk_a-f_Nk-
    alphaj*dmk))
                if alphaj_new<0.1*alphaj or alphaj_new>0.9*alphaj:

                    alphaj_new=alphaj/2
                    alphaj= alphaj_new


                else:

                    alphaj=alphaj/2
```

Bibliography

[1] https://www.aitude.com/supervised-vs-unsupervised-vs-reinforcement/.

[2] https://www.javatpoint.com/supervised-machine-learning.

[3] https://www.researchgate.net/figure/Figure-6-3-Global-and-local-optimum-solutions-Adapted-from-130_fig31_294872230.

[4] https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/.

[5] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html.

[6] https://www.causality.inf.ethz.ch/data/CINA.html.

[7] https://medium.com/@sunside/quadratic-interpolation-given-two-points-and-one-derivative-61837bfa1e05.

[8] https://www.geeksforgeeks.org/understanding-logistic-regression/.

[9] R. Asmundis, D. Di Serafino, F. Riccio, and G. Toraldo, On spectral properties of steepest descent methods, IMA Journal of Numerical Analysis, 33 (2013).

[10] J. Barzilai and J. Borwein, Two point step gradient methods, IMA Journal of Numerical Analysis, 8 (1988), pp. 141–148.

[11] S. Bellavia, N. Krejić, N. K. Jerinkić, and M. Raydan, Slises: Subsampled line search spectral gradient method for finite sums, 2024.

[12] S. Bellavia, N. Krklec Jerinkic, and G. Malaspina, Subsampled nonmonotone spectral gradient methods, Communications in Applied and Industrial Mathematics, 11 (2020), pp. 19–34.

[13] E. G. Birgin, J. M. Martínez, and M. Raydan, Spectral projected gradient methods: Review and perspectives, Journal of Statistical Software, 60 (2014), p. 1–21.

[14] L. Bottou, F. E. Curtis, and J. Nocedal, Optimization methods for large-scale machine learning, SIAM Review, 60 (2018), pp. 223–311.

[15] Y.-H. Dai, W. Hager, K. Schittkowski, and H. Zhang, The cyclic barzilai–borwein method for unconstrained optimization, IMA Journal of Numerical Analysis, 26 (2005).

[16] D. di Serafino, V. Ruggiero, G. Toraldo, and L. Zanni, On the steplength selection in gradient methods for unconstrained optimization, Applied Mathematics and Computation, 318 (2018), pp. 176–195.

[17] A. Friedlander, N. Krejić, and N. Krklec Jerinkić, Fundamentals of Numerical Optimization, University of Novi Sad Faculty of Sciences, 2019.

[18] D. Jakovetic, N. Krejić, and N. Krklec Jerinkic, Exact spectral-like gradient method for distributed optimization, Computational Optimization and Applications, 74 (2019).

[19] N. Krejić and N. K. Jerinkić, Spectral projected gradient method for stochastic optimization, Journal of Global Optimization, 73 (2019), pp. 59–81.

[20] M. Raydan, On the barzilai and borwein choice of steplength for the gradient method, IMA Journal of Numerical Analysis, 13 (1993), pp. 321–326.

[21] C. Tan, S. Ma, Y.-H. Dai, and Y. Qian, Barzilai-borwein step size for stochastic gradient descent, 2016.

[22] A. Tsanas, M. A. Little, C. Fox, and L. O. Ramig, Objective automatic assessment of rehabilitative speech treatment in parkinson's disease, IEEE Trans. Neural Syst. Rehabil. Eng., 22 (2014), pp. 181–190.

[23] Z. Yang, C. Wang, Z. Zhang, and J. Li, Random barzilai-borwein step size for mini-batch algorithms, Engineering Applications of Artificial Intelligence, 72 (2018), pp. 124–135.

Biography

Isidora Vuković was born on the 26th of March, 1999 in Priština, Serbia. Due to the bombing at the beginning of the year 2000, she came to Novi Sad, where she still lives. She attended elementary school ”Svetozar Marković Toza” and grammar school ”Jovan Jovanović Zmaj”. In 2017 she started her bachelor degree studies in Financial Mathematics at Faculty of Sciences, University of Novi Sad, which she finished in 2021 with the GPA 8.08. In the same year she continued with her Master studies of Data Science at the same faculty and passed all exams in 2023 with the GPA of 9.06.

UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI FAKULTET
KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj:
RBR
Identifikacioni broj:
IBR
Tip dokumentacije: Monografska dokumentacija
TD
Tip zapisa: Tekstualni štampani materijal
TZ
Vrsta rada: Master rad
VR
Autor: Isidora Vuković
AU
Mentor: Dr Nataša Krklec Jerinkić
MN
Naslov rada: Primena stohastičkog metoda spektralnog gradijenta na probleme mašinskog učenja
NR
Jezik publikacije: Engleski
JP
Jezik izvoda: Engleski
JI
Zemlja publikovanja: Republika Srbija
ZP
Uže geografsko područje: Vojvodina
UGP
Godina: 2024.
GO
Izdavač: Autorski reprint
IZ
Mesto i adresa: Novi Sad, Trg Dositeja Obradovića 4
MA
Fizički opis rada: 7 poglavlja, 45 strana, 23 citata, 27 slika
FO
Naučna oblast: Matematika
NO
Naučna disciplina: Primenjena matematika
ND
Ključne reči: mašinsko učenje, numerička optimizacija, stohastička optimizacija, metoda spektralnog gradijenta, spektralni koeficijent, lokalne funkcije gubitka, SLiSeS algoritam, linijsko pretraživanje
KR
Univerzalna decimalna klasifikacija:
UDK
Čuva se: Biblioteka Departmana za matematiku i informatiku, Prirodno-matematičkog fakulteta, u Novom Sadu

ČU
Važna napomena:
VN
Izvod: Glavni cilj ove teze je ispitivanje ponašanja SLiSeS algoritma u različitim scenarijima. Fokus je na analizi uticaja različitih parametara na performanse algoritma. Pored toga, istraženo je kako se SLiSeS algoritam ponaša u situacijama sa različitim tipovima podataka i različitim tipovima problema optimizacije. Cilj je da se stekne dublje razumevanje kako SLiSeS funkcioniše u različitim situacijama i kako se može najefikasnije primeniti u praksi. U cilju što boljeg razumevanja prvo je data teorijska osnova svih komponenata SLiSeS algoritma, a zatim su prikazani numerički rezultati i dati izvedeni zaključci.
IZ
Datum prihvatanja teme od strane NN veća:
DP
Datum odbrane:
DO
Članovi komisije:
KO
Predsednik: Dr Sanja Rapajić, redovni profesor, Prirodno-matematički fakultet, Novi Sad
Mentor: Dr Nataša Krklec Jerinkić, vanredni profesor, Prirodno-matematički fakultet, Novi Sad
Član: Dr Dušan Jakovetić, vanredni profesor, Prirodno-matematički fakultet, Novi Sad

UNIVERSITY OF NOVI SAD
FACULTY OF SCIENCE
KEY WORDS DOCUMENTATION

Accession number:
ANO
Identification number:
INO
Document type: Monograph type
DT
Type of record: Printed text
TR
Contents code: Master thesis
CC
Author: Isidora Vuković
AU
Mentor: Dr Nataša Krklec Jerinkić
MN
Title: Application of a Stochastic Spectral Gradient Method on Machine Learning Problems
TL
Language of text: English
LT
Language of abstract: English
LA
Country of publication: Republic of Serbia
CP
Locality of publication: Vojvodina
LP
Publication year: 2024.
PY
Publisher: Author's reprint
PU
Publ. place: Novi Sad, Trg Dositeja Obradovića 4
PP
Physical description: 7 chapters, 45 pages, 23 references, 27 figures
PD
Scientific field: Mathematics
SF
Scientific discipline: Applied mathematics
SD
Key words: machine learning, numerical optimization, stochastic optimization, spectral gradient method, spectral coefficient, local loss functions, SLiSeS algorithm, line search
KW
Universal decimal classification:
UDC
Holding data: Department of Mathematics and Informatics' Library, Faculty of Sciences, Novi Sad

HD
Note:
N
Abstract: The main goal of this thesis is to examine the behavior of the SLiSeS algorithm in various scenarios. The focus is on analyzing the impact of different parameters on the algorithm's performance. Additionally, the behavior of the SLiSeS algorithm in situations with different types of data and different types of optimization problems was investigated. The aim is to gain a deeper understanding of how the SLiSeS operates in different situations and how it can be most effectively applied in practice. In order to better understand this, first the theoretical basis of all components of the SLiSeS algorithm is given, followed by the presentation of numerical results and derived conclusions.
AB
Accepted by the Scientific Board on:
ASB
Defended:
DE
Thesis defend board:
DB