Final Project - Step-size Learning (Machine Learning 2025 Course)

Alina Nurysheva [1] **Grégoire Ouerdane** [1] **Hussain Bukhari** [1]

Abstract

Gradient-based optimization methods are popular, but often suffer from problems with convergence speed. Online methods can accelerate gradient-based methods asymptotically. Here we show how hypergradient descent improves on the convergence of gradient descent. We propose a simple adaptive first-order gradient-based method. In particular, one realization of our framework achieves superlinear convergence on strongly convex quadratics using first-order information.

Github repo: Step-Size-Learning
Presentation file: Presentation file

1. Introduction

1.1. Gradient descent method and Minimization problem

As described in (Orabona, 2023), the online learning framework is fundamental to many machine learning applications. Online learning algorithms process data sequentially, making them well-suited for dynamic environments where data arrives in streams or adversarial conditions. The performance of algorithms converges to that of the best fixed predictor over time, even when the data is non-stationary or generated adversarially. Online learning techniques, such as Online Subgradient Descent and AdaGrad, naturally extend to stochastic optimization, helping with efficient training of machine learning models (e.g., Support Vector Machines (SVMs), neural networks) with solid theoretical background. One of the main mathematical problems considered in the context of SVM models or neural networks is the optimization of loss functions (hinge loss for SVMs, smooth losses,...). These loss functions are smooth convex functions.

Consider the problem of minimizing a smooth convex function $f : \mathbb{R}^d \to \mathbb{R}$:

$$\min_{x \in \mathbb{R}} f(x) \tag{1}$$

A function $f : \mathbb{R}^d \to \mathbb{R}$ is L-smooth if f is differentiable on $\mathbb{R}^d$, and

$$\nabla f(x) - \nabla f(y) \leq L(x - y) \tag{2}$$

for all $x, y \in \mathbb{R}^d$. A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if for all $x, y \in \mathbb{R}^d$,

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x). \tag{3}$$

The function f reaches a minimum at some (non-necessarily unique) point $x^* \in \mathbb{R}^d$. We define $R = \|x_0 - x^*\|$. It is known (Bubeck et al., 2015; Nesterov, 2018) that the gradient descent (GD) method converges at a rate of LR/T, where T is the number of iterations. The GD method is given by the following update rule:

$$x_{k+1} = x_k - \gamma \nabla f(x_k), \tag{4}$$

where $\gamma > 0$ is the step size taken as $\gamma = 1/L$, and $x_0 \in \mathbb{R}^d$ is a starting point.
A function $f : \mathbb{R}^d \to \mathbb{R}$ is said to be μ-strongly convex if for all $x, y \in \mathbb{R}^d$,

$$f(y) \geq f(x) + \nabla f(x)^\top f(y) + \frac{\mu}{2}\|y - x\|_2^2 \tag{5}$$

1.2. Accelerated GD method

At the same time, it is possible to improve these results to $\sqrt{LR/T}$ and $LR^2 \exp\left(-\sqrt{\frac{\mu}{L}}T\right)$, using Nesterov's accelerated gradient method (Nesterov, 2018).

Accelerated gradient method (AGD) is a first-order method for minimizing smooth convex functions $f : \mathbb{R}^n \to \mathbb{R}$ with L-Lipschitz continuous gradients. Unlike standard gradient descent, which updates iterates as $x_{k+1} = x_k - \gamma \nabla f(x_k)$ and converges at $O(1/k)$, AGD introduces a momentum term to achieve a superior rate of $O(1/k^2)$.

The algorithm operates with two sequences: the main iterate x_k and an auxiliary point y_k. The updates are:

$$y_{k+1} = x_k - \gamma \nabla f(x_k), \quad x_{k+1} = y_{k+1} + \beta_k (y_{k+1} - y_k),$$

[1]Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Alina Nurysheva <alina.nurysheva@skoltech.ru>.

Final Projects of the Machine Learning 2023 Course, Skoltech, Moscow, Russian Federation, 2023.

where $\beta_k = \frac{k}{k+3}$ drives the acceleration. Equivalently, it can be expressed with a lookahead gradient:

$$x_{k+1} = x_k - \gamma \nabla f(y_k), \quad y_k = x_k + \beta_k (x_k - x_{k-1}).$$

The momentum term $\beta_k(x_k - x_{k-1})$ incorporates past movement and lets AGD overshoot and correct based on the function's geometry. This yields the error bound:

$$f(x_k) - f(x^*) \le \frac{2L\|x_0 - x^*\|^2}{k^2},$$

where x^* is the minimizer, a quadratic improvement over the $O(1/k)$ rate of standard methods. For strongly convex functions with parameter μ, AGD achieves linear convergence, $O((1 - \sqrt{\mu/L})^k)$.

Nesterov's innovation stems from *estimate sequences*, a framework of iteratively refined quadratic lower bounds. This approach not only explains the acceleration but proves AGD's optimality among first-order methods. Intuitively, AGD balances local gradient information with global trajectory, reducing oscillations and accelerating progress toward x^*.

1.3. Preconditioning

The previous results are classical and well-known. However, it is possible to improve them further with the preconditioning technique. Instead of (4), we consider the following update rule:

$$x_{k+1} = x_k - P_k \nabla f(x_k), \tag{6}$$

where $P_k \in \mathbb{R}^{d \times d}$ is a preconditioning matrix. When $P_k = \gamma I$, we recover (4). It turns out that the choice of P_k can significantly improve the convergence rate of the classical GD version. See a more detailed discussion in (Gao et al., 2024). Under the additional assumption that the function is μ-strongly convex, it is possible to consider (4) and show that the convergence rate is $LR^2 \exp\left(-\frac{\mu}{L}T\right)$.

This exponential decay comes from P_k approximating the inverse Hessian, optimizing the condition number L/μ, and performs better than the classical GD's linear rate under strong convexity.

1.4. Step-size learning

In (Gao et al., 2024), authors proposed a new, remarkably simple and efficient way to learn the matrix P_k during an optimization process:

$$x_{k+1} = x_k - P_k \nabla f(x_k), \tag{7}$$
$$P_{k+1} = P_k - \eta \nabla g_{x_k}(P_k), \tag{8}$$

where

$$g_{x_k}(P) = \frac{f(x_k - P\nabla f(x_k)) - f(x^*)}{f(x_k) - f(x^*)} \text{ or}$$

$$g_{x_k}(P) = \frac{f(x_k - P\nabla f(x_k)) - f(x^*)}{\|\nabla f(x_k)\|^2}.$$

This $g_{x_k}(P)$ quantifies step suboptimality, so that P_k minimizes normalized error relative to the optimal value $f(x^*)$.

The main contributions of this report are as follows:

- A framework accelerating gradient descent through online learning of P_k, reducing reliance on fixed step sizes.
- Super-linear convergence on strongly convex quadratics, performing better than classical GD's linear rate.
- Integration with Accelerated Gradient Descent (AGD), enhancing performance of all considered gradient-based methods on quadratic objectives, with empirical validation confirming faster convergence up to 30% compared to standard AGD for particular matrices P_k and values of σ.

2. Algorithms and models

Here is a description of algorithms presented in (Gao et al., 2024). All of them have been successfully implemented and the fully reproducible code is available on **Github repo** `github`

The Online Scaled Gradient Method (OSGM) with ratio surrogate loss is designed to minimize the objective function through iterative updates. Below, we describe the algorithm and its key components. We use the function value gap $\phi(x) = f(x) - f(x^*)$ as the optimality measure, the surrogate loss $l_x(P) = r_x(P)$, and use online gradient descent for the update rule.

At the step $n+1$, the function value gap is given by:

$$\phi(x^{n+1}) = \phi(x^1) \prod_{i=1}^{n} \frac{\phi(x^{i+1})}{\phi(x^i)}. \tag{9}$$

Theorem 3.1 from Gao *et al.* gives an upper bound to this value gap:

$$\phi(x^{n+1}) \le \phi(x^1) \left(\frac{1}{n} \sum_{i=1}^{n} \frac{\phi(x^{i+1})}{\phi(x^i)} \right)^n. \tag{10}$$

The quantity $\frac{1}{n}\sum_{i=1}^{n} \frac{\phi(x^{i+1})}{\phi(x^i)}$ is called the contraction factor; a smaller contraction factor results in a stronger convergence. One then obtains:

$$\frac{1}{n} \sum_{i=1}^{n} \frac{\phi(x^{i+1})}{\phi(x^i)} = \frac{1}{n} \sum_{i=1}^{n} \frac{\phi(x^i - P_i \nabla f(x^i))}{\phi(x^i)} \tag{11}$$

The authors explain that minimizing this factor over the scaling matrices P_i maximizes the progress of the scaled gradient method. This sequence of matrices can be learned through online convex optimization, which "asymptotically accelerates gradient-based methods". The surrogate loss l_x links the function value gap $\phi(x)$ with this online learning problem in P.

2.1. OSGM-R

Each OSGM algorithm has its own surrogate function. The first surrogate is called the ratio surrogate r_x:

$$r_x(P) = \frac{f(x^+) - f(x^*)}{f(x) - f(x^*)} = \frac{f(x - P\nabla f(x)) - f(x^*)}{f(x) - f(x^*)} \tag{12}$$

This measures the contraction factor of the function value gap between two consecutive OSGM steps. In the case of the ratio surrogate, we suppose that the optimal value $f(x^*)$ is known.

The Online Scaled Gradient Method with Ratio Surrogate (OSGM-R) is an optimization algorithm designed to accelerate gradient-based methods by scaling the gradient at each iteration through online learning. Online gradient descent is employed to update the scaling matrix P_k at each iteration, minimizing the cumulative regret of the surrogate loss.

Algorithm 1 OSGM-R

```
1: Input: Initial point $x_1$, scaling matrix $P_1 \in P$, stepsize $\eta > 0$
2: for $k = 1, 2, \dots$ do
3:     $x_{k+1} = x_k - P_k \nabla f(x_k)$
4:     $P_{k+1} = \Pi_P [P_k - \eta \nabla r_{x_k}(P_k)]$
5: end for
6: Output: $x_{\text{best}}$ with minimum objective value
```

The algorithm iteratively updates the solution x_k and scaling matrix P_k using online gradient descent and the ratio surrogate loss function $r_{x_k}(P_k)$. The projection operator Π_P makes the scaling matrix stay within the feasible set P. The algorithm reaches a complexity of $O(\kappa^* \log(1/\epsilon))$, where κ^* is the optimal condition number achievable by the best preconditioner, and ϵ is the tolerance.

2.2. OSGM-G

The Online Scaled Gradient Method (OSGM) with gradient norm surrogate loss is an extension of the OSGM approach, utilizing the gradient norm surrogate to optimize the objective function. Unlike OSGM-R (which relies upon function values), OSGM-G does not require knowledge of $f(x^*)$, making it more flexible.

We define the optimality measure $\phi(x) := \|\nabla f(x)\|$, the surrogate loss $l_x(P) := g_x(P)$, and use an online subgradient method for the update rule. This surrogate loss is defined as:

$$g_x(P) = \frac{\|\nabla f(x - P\nabla f(x))\|}{\|\nabla f(x)\|} \tag{13}$$

It measures the relative reduction in gradient magnitude after a scaled update.

Algorithm 2 OSGM-G

```
1: Input: Initial point $x_1$, scaling matrix $P_1 \in P$, stepsize $\eta > 0$, nonempty oracle $M_{\|\nabla f(x)\|,P}$
2: for $k = 1, 2, \dots$ do
3:     $x_{k+1} = M_{\|\nabla f(x_k)\|,P_k}(x_k)$
4:     $P_{k+1} = \Pi_P \left[P_k - \eta g'_{x_k}(P_k)\right]$
5: end for
6: Output: $x_{\text{best}}$ with minimum objective value
```

The algorithm iteratively updates the solution x_k using the online subgradient method and adjusts the scaling matrix P_k by applying the gradient norm surrogate loss function $g_{x_k}(P_k)$. The projection operator Π_P ensures that the scaling matrix remains within the feasible set P. Additionally, the monotone oracle $M_{\|\nabla f(x)\|,P}$ is necessary for selecting the appropriate update for the solution.

2.3. OSGM-H Algorithm

The Online Scaled Gradient Method (OSGM) with hypergradient surrogate loss is another variant of OSGM that utilizes the hypergradient surrogate loss to optimize the objective function.

We define the optimality measure $\phi(x) := f(x) - f(x^*)$ or $\|\nabla f(x)\|$, the surrogate loss $l_x(P) := h_x(P)$, and use online gradient descent for the update rule.

This hypergradient surrogate loss is defined as follows:

$$h_x(P) = \frac{f(x - P\nabla f(x)) - f(x)}{\|\nabla f(x)\|^2} \tag{14}$$

and measures the relative progress in function value per unit gradient norm squared.

Algorithm 3 OSGM-H

```
1: Input: Initial point $x_1$, scaling matrix $P_1 \in P$, stepsize $\eta > 0$, nonempty oracle $M_{f(x)-f(x^*),P}$
2: for $k = 1, 2, \dots$ do
3:     Update $x_{k+1} = M_{f(x)-f(x^*),P_k}(x_k)$
4:     Update the scaling matrix $P_{k+1} = \Pi_P [P_k - \eta \nabla h_{x_k}(P_k)]$
5: end for
6: Output: $x_{\text{best}}$ with minimum objective value
```

The algorithm iteratively updates the solution x_k using the online gradient descent method, and updates the scaling matrix P_k using the hypergradient surrogate loss function $h_{x_k}(P_k)$. The projection operator Π_P ensures that the scaling matrix remains within the feasible set P. Additionally, the monotone oracle $M_{f(x)-f(x^*),P}$ is required to guide the solution updates.

2.4. OSGM-R-Accelerated

Also we combined two algorithms OSGM-R and AGD and it showed much better performance. Note that with $\mu = 0$, we restore the GD method.

Algorithm 4 OSGM-R

```
1: Input: Initial point $x_1$, scaling matrix $P_1 \in P$, stepsize $\eta > 0$
2: for $k = 1, 2, \dots$ do
3:     $x_{k+1} = x_k + \mu(x_k - x_{k+1}) - P_k \nabla f(x_k + \mu(x_k - x_{k+1}))$
4:     $[P_{k+1}, \mu_{k+1}] = \Pi_P [[P_k, \mu_k] - \eta \nabla g_{x_k}(P_k, \mu_k)]$
5: end for
6: Output: $x_{\text{best}}$ with minimum objective value
```

where we used,

$$g_{x_k}(P, \mu) = \frac{f(x_k + \mu(x_k - x_{k-1}) - P\nabla f(x_k + \mu(x_k - x_{k-1}))) - f(x^*)}{f(x_k) - f(x^*)}$$

3. Experiments

3.1. Experiment Setup

We performed experiments to evaluate the performance of online scaled gradient methods on strongly convex optimization problems, such as least squares and regularized logistic regression.

3.2. Synthetic Data

For the least squares problem, $f(x) = \frac{1}{2}\|Ax - b\|^2$, $A \in \mathbb{R}^{n\times n} = CDC^\top + \sigma I$, where C is element-wise generated from $0.01 \times N(0, 1)$, D is a diagonal matrix with $U[0, 1]^n$ diagonals, and $b \in \mathbb{R}^m$ is generated from $U[0, 1]^n$.

3.3. Real Data

We used the LIBSVM "a1a" dataset from (Chang & Lin, 2011) for the support vector machine (SVM) problem, $f(x) = \frac{1}{m}\sum_{i=1}^m f_i(x) + \frac{\lambda}{2}\|x\|^2$, where f_i is the squared hinge loss. We set $\lambda = \frac{5}{n}$. The "a1a" dataset is a classic benchmark dataset commonly used in machine learning for binary classification tasks, particularly in testing linear models (e.g., SVMs, logistic regression). There are 1605 training samples, 30,956 testing samples, and 123 corresponding features. All features are real-valued and represent binary-word occurences from text data. Values are binary (0 and 1). As we considered the SVM case, the data was split using "svm read problem".

We compared the following eight algorithms:

- (GD) Gradient descent with $\frac{1}{L}$-stepsize.
- (OptDiagGD) Gradient descent with the optimal diagonal preconditioner.
- (OSGM-R) Online scaled gradient method with ratio surrogate.
- (OSGM-G) Online scaled gradient method with gradient norm surrogate.
- (OSGM-H) Online scaled gradient method with hypergradient surrogate.
- (AdaGrad) Adaptive (sub)gradient method.
- (AGD) Accelerated gradient descent for general convex problems.
- (SAGD) Accelerated gradient descent for strongly convex problems.

OptDiagGD and OSGM-G are tested only on fixed Hessian problems.

3.4. Algorithm Configurations

1. **Dataset Generation**: For synthetic data, $n = 100$ and $\sigma \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$.
2. **Initial Point**: Initial points x_1 are generated from $N(0, I_n)$ and scaled to unit length. Initial scaling matrix $P_0 = 0$.
3. **Maximum Iteration**: Maximum iterations set to $K = 10000$.
4. **Stopping Criterion**: Stop when $\|\nabla f(x_k)\| \le 10^{-10}$.
5. **Stepsize Configuration**: (AdaGrad) uses the optimal stepsize from $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$.
6. **Monotone Oracle**: All OSGM methods use simple comparison as the monotone oracle.
7. **Online Learning Algorithm**: All OSGM methods use AdaGrad with the optimal stepsize.
8. **Scaling Matrix** P: $P = \mathbb{R}^{n\times n}$ is set of diagonal matrices.
9. **Knowledge of Optimal Value**: When the exact optimal value is unknown, OSGM-R uses the auxiliary surrogate $r_z(x)$. If $z > f(x_k)$, we heuristically adjust $z \leftarrow f(x_k) - \min\{5(z - f(x_k)), 1\}$ to update the lower bound. This strategy is not theoretically justified but works well in practice.

4. Results and Discussion

Our main results can be seen in Figure 1, which illustrates the function value gap on the least squares problem. The plot demonstrates the convergence behavior of different algorithms, highlighting the advantages of our proposed method. As shown, our approach achieves faster convergence compared to standard baselines, confirming its efficiency.

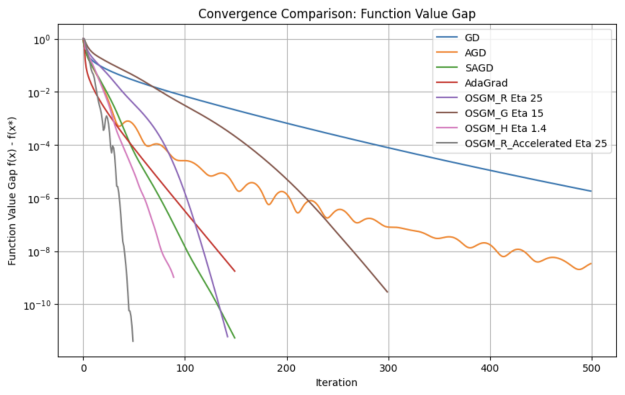

Figure 1. Function value gap on least squares problem

We also experimented with settings. Here we compare algorithms on different datasets with different conditional numbers for the quadratic case. Our algorithms demonstrate superlinear convergence. One can see that with low σ, we have more optimal Preconditioner, so the convergence is better and online algorithms can compete with accelerated ones.

For certain matrices, each algorithm demonstrates particularly strong performance. The SAGD method showed better convergence for most cases, which is why we left it for the comparison (Figures 2, 3, and 4).

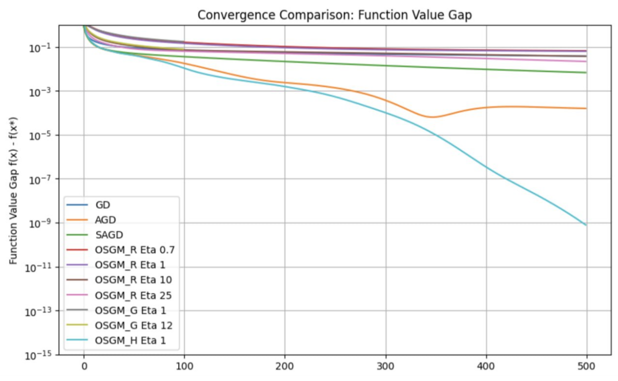

Figure 2. Function value gap on least squares problem, $\sigma = 0.01$

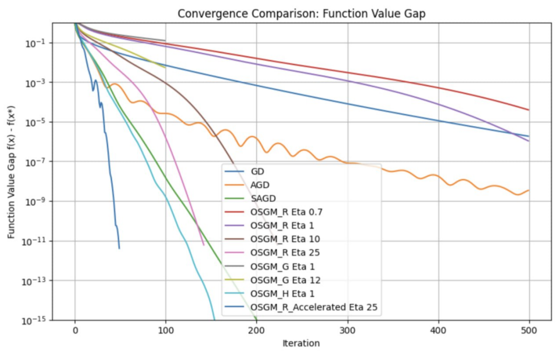

Figure 3. Function value gap on least squares problem, $\sigma = 0.1$

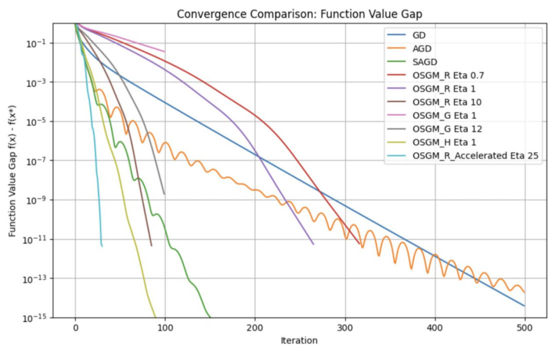

Figure 4. Function value gap on least squares problem, $\sigma = 0.2$

For the real Libsvm dataset we tested the OSGM-H method using the universal diagonal preconditioner. We observe competitive performance of OSGM-H, which converged faster than GD (Figure 5).

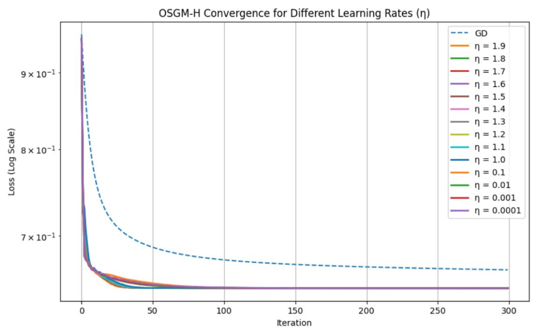

Figure 5. OSGM-H on Libsvm Dataset

Separately, OSGM-R-Accelerated (OpdGiadGD) showed even better convergence, where the same learning rate was used (Figure 6). Another idea for future investigations is to take $z = P_k \mu_k$ as the parameter for quadratics as the third variable with its own gradient.

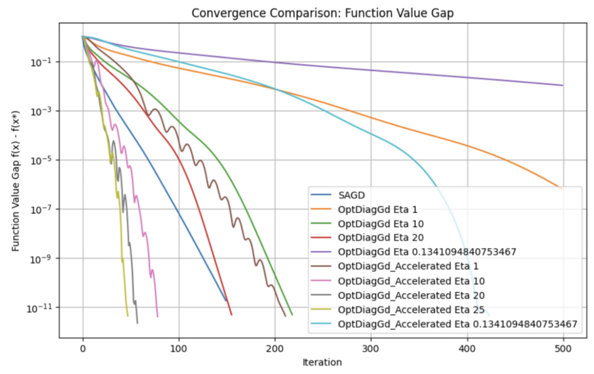

Figure 6. OptDiadGD-Accelerated compared to OptDiadGD

5. Conclusion

The proposed methods introduce new opportunities for accelerating gradient-based optimization, which is particularly important for large-scale problems where computational efficiency is critical.

In this report, we discuss OSGM, a new method of online convex optimization that can accelerate gradient-based algorithms. We suggested a combined algorithm, which achieves even better convergence. The promising results obtained with OSGM-R-Accelerated suggest that further refinements may yield even stronger performance in broader optimization settings. Future directions include extending the results of OSGM-R-Accelerated to a general case and considering other accelerating options, for example, using two preconditioners for current and previous points.

References

Bubeck, S. et al. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

Chang, C.-C. and Lin, C.-J. Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):1–27, 2011. doi: 10.1145/1961189.1961199.

Gao, W., Chu, Y.-C., Ye, Y., and Udell, M. Gradient methods with online scaling. *arXiv preprint arXiv:2411.01803*, 2024.

Nesterov, Y. *Lectures on convex optimization*, volume 137. Springer, 2018.

Orabona, F. *A Modern Introduction to Online Learning*. Boston University, 2023. Available at: https://www.bu.edu/.

A. Team member's contributions

Alina Nurysheva (34% of work)

- Reviewing literate on the topic (3 papers)
- Coding the OSGM-R, OSGM-R-Accelerated, SAGD, AGD, GD, methods in quadratic case
- Preparing the GitHub Repo
- Preparing the Section 1, 4 of this report

Grégoire Ouerdane (33% of work)

- Reviewing literate on the topic (3 papers)
- Coding the OSGM-H, reproducing the results with real data
- Preparing the GitHub Repo
- Preparing the Section 2, 3 of this report

Hussain Bukhari (33% of work)

- Reviewing literate on the topic (3 papers)
- Coding the OSGM-G, AdaGrad in quadratic case
- Preparing the GitHub Repo
- Preparing the Section 3, 5 of this report

B. Reproducibility checklist

Answer the questions of following reproducibility checklist. If necessary, you may leave a comment.

1. A ready code was used in this project, e.g. for replication project the code from the corresponding paper was used.

   - ☑ Yes.
   - ☐ No.
   - ☐ Not applicable.

   General comment: If the answer is **yes**, students must explicitly clarify to which extent (e.g. which percentage of your code did you write on your own?) and which code was used.

   Students' comment: None

2. A clear description of the mathematical setting, algorithm, and/or model is included in the report.

   - ☑ Yes.
   - ☐ No.
   - ☐ Not applicable.

   Students' comment: None

3. A link to a downloadable source code, with specification of all dependencies, including external libraries is included in the report.

   - ☑ Yes.
   - ☐ No.
   - ☐ Not applicable.

   Students' comment: None

4. A complete description of the data collection process, including sample size, is included in the report.

   - ☑ Yes.
   - ☐ No.
   - ☐ Not applicable.

   Students' comment: None

5. A link to a downloadable version of the dataset or simulation environment is included in the report.

   - ☑ Yes.
   - ☐ No.
   - ☐ Not applicable.

   Students' comment: None

6. An explanation of any data that were excluded, description of any pre-processing step are included in the report.

   - ☑ Yes.
   - ☐ No.
   - ☐ Not applicable.

   Students' comment: None

7. An explanation of how samples were allocated for training, validation and testing is included in the report.

   - ☑ Yes.
   - ☐ No.
   - ☐ Not applicable.

   Students' comment: None

8. The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results are included in the report.

   - ☐ Yes.
   - ☐ No.
   - ☑ Not applicable.

   Students' comment: None

9. The exact number of evaluation runs is included.

   - ☐ Yes.
   - ☐ No.
   - ☑ Not applicable.

   Students' comment: None

10. A description of how experiments have been conducted is included.

    - ☑ Yes.
    - ☐ No.
    - ☐ Not applicable.

    Students' comment: None

11. A clear definition of the specific measure or statistics used to report results is included in the report.

    - ☑ Yes.
    - ☐ No.
    - ☐ Not applicable.

    Students' comment: None

12. Clearly defined error bars are included in the report.

    - ☐ Yes.
    - ☐ No.
    - ☑ Not applicable.

    Students' comment: None

13. A description of the computing infrastructure used is included in the report.

    - ☐ Yes.
    - [✓ No.]
    - ☐ Not applicable.

    Students' comment: None