UNIVERSITÀ DEGLI STUDI DI PISA

DIPARTIMENTO DI INFORMATICA

Computational Mathematics for Learning and Data Analysis

Support Vector Regression using Smoothed Gradient methods

Student:
Merialdo Margherita
Mat. 690870

Student:
Faella Michela
Mat. 694416

ACADEMIC YEAR 2024/2025

Track Project 8

(M) is a SVR (support vector regression)-type approach of your choice (in particular, with one or more kernels of your choice).
(A1) is an algorithm of the class of smoothed gradient methods, cf. also here, applied to either the primal or the dual formulation of the SVR.
(A2) is a general-purpose solver applied to an appropriate formulation of the problem.
No off-the-shelf solvers allowed, save of course for (A2).

Contents

Chapter 1

Introduction

The purpose of our project is to develop a Support Vector Machine for solving **Regression** tasks. Support Vector Regression (SVR) searches for a function (or hypothesis) that approximates the target value while maintaining a margin tolerance. So, the model is initially defined as:

$$h(x) = w^T x + b \tag{1.1}$$

Then we apply the **Linear Basis Expansion** (LBE) that allows us to estimate the target function using a linear expansion of non-linear functions. This transforms our model into:

$$h(x) = w^T \phi(x) \tag{1.2}$$

where $\phi(x)$ represents a mapping to a higher-dimensional feature space.

An important role in the SVR is covered by the hyperparameter C, which is tuned to generate a correct trade-off between the overfitting (an excess of precision) and the underfitting (and an excess of generalization). The primary formulation of the SVR optimization problem is defined as:

$$\min_{w,\xi,\xi'} \frac{1}{2}||w||^2 + C \sum_{i=1}^{N}(\xi_i + \xi_i') \tag{1.3}$$

subject to the constraints:

$$\forall i = 1, \dots, N \begin{cases} d_i - \mathbf{w}^T \Phi(\mathbf{x}_i) \leq \epsilon + \xi_i \\ \mathbf{w}^T \Phi(\mathbf{x}_i) - d_i \leq \epsilon + \xi_i' \\ \xi_i \geq 0 \\ \xi_i' \geq 0 \end{cases} \tag{1.4}$$

where:

- $||\mathbf{w}||^2$ represents the complexity of the model that we aim to minimize,
- $C \sum(\xi_i + \xi_i')$ represents the regularization term, which controls the trade-off between complexity and error tolerance,
- ϵ defines the insensitivity zone, ensuring that small deviations are ignored,
- ξ_i, ξ_i' are slack variables that allow for soft-margin violations.

The most common loss function is the ϵ-insensitive:

$$L(y_i - f(x_i)) = max(0, |y_i - f(x_i)| - \epsilon) \tag{1.5}$$

Now that we have all the necessary ingredients, we can define the implementation chosen for each point of our project.

1.1 Kernel Implementation

To achieve the point **M**, we implement three different kernel functions and evaluate their performance to determine the most suitable for our datasets[1]. In particular, we implement:

- **Linear Kernel** = The linear kernel is the simplest kernel function and is defined as:

  $$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j \tag{1.6}$$

  This kernel is effective when the data are linearly separable in the input space.

- **Polynomial Kernel** = The polynomial kernel allows for the modelling of more complex relationships by introducing a polynomial transformation of the input features:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j + c)^p \tag{1.7}$$

  where p is the degree of polynomial and c is a constant that controls the influence of higher-order terms.

- **Radial Basis Function (RBF) Kernel** = The RBF kernel, also known as the Gaussian kernel, is widely used due to its ability to capture complex patterns. It is defined as:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right) \tag{1.8}$$

  where γ is a hyperparameter that controls the width of the Gaussian function, influencing how much a single training example impacts the decision boundary.

1.2 Dual Formulation

To solve the point **(A1)**, we chose to approach the problem using the **dual formulation** of SVR. The dual form allows us to efficiently incorporate kernel functions and solve the optimization problem in a higher-dimensional feature space. To transform the primal formulation into the dual one, we have to introduce the Lagrange multiplier.

$$\begin{aligned}\mathcal{L}(w, \xi, \xi', \alpha, \alpha', \mu, \mu') =& \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i') \\ & - \sum_{i=1}^{N} \alpha_i(\epsilon + \xi_i - d_i + \mathbf{w}^T\Phi(\mathbf{x}_i)) \\ & - \sum_{i=1}^{N} \alpha_i'(\epsilon + \xi_i' + d_i - \mathbf{w}^T\Phi(\mathbf{x}_i)) \\ & - \sum_{i=1}^{N} \mu_i\xi_i - \sum_{i=1}^{N} \mu_i'\xi_i'\end{aligned} \tag{1.9}$$

where:

- $\alpha_i, \alpha_i' \geq 0$ are for the prediction constraints.
- $\mu_i, \mu_i' \geq 0$ are to ensure the non-negativity of the slack variables ξ_i, ξ_i'.

Setting the derivatives with respect to w, ξ and ξ' to zero, we obtain the **dual form**:

$$\max_{\alpha,\alpha'} Q(\boldsymbol{\alpha}, \boldsymbol{\alpha}') \tag{1.10}$$

$$Q(\boldsymbol{\alpha}, \boldsymbol{\alpha}') = \sum_{i=1}^{N} d_i(\alpha_i - \alpha_i') - \epsilon \sum_{i=1}^{N}(\alpha_i + \alpha_i') - \frac{1}{2}\sum_{(i,j)=1}^{N}(\alpha_i - \alpha_i')(\alpha_j - \alpha_j')k(\mathbf{x}_i, \mathbf{x}_j) \tag{1.11}$$

subject to the constraints:

$$0 \leq \alpha_i, \alpha_i' \leq C, \quad \forall i = 1, \ldots, N, \tag{1.12}$$

$$\sum_{i=1}^{N}(\alpha_i - \alpha_i') = 0. \tag{1.13}$$

where:

- α_i and α_i' are the Lagrange multipliers,
- C is the regularization parameter that controls the trade-off between margin size and error,
- $K(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel function used to map data into a higher-dimensional space.

Now we define a new variable β_i as

$$\beta_i = \alpha_i - \alpha_i' \tag{1.14}$$

and we notice that the constraint (1.13) becomes:

$$\sum_{i=1}^{N} \beta_i = 0$$

and the quadratic term in the Equation (1.11) becomes:

$$(\alpha_i - \alpha_i')(\alpha_j - \alpha_j') = \beta_i \beta_j \tag{1.15}$$

It follows that the linear term:

$$\sum_{i=1}^{N} d_i(\alpha_i - \alpha_i') - \epsilon \sum_{i=1}^{N} (\alpha_i + \alpha_i') \tag{1.16}$$

can be rewritten by separating the linear part as:

$$\sum_{i=1}^{N} d_i \beta_i - \epsilon \sum_{i=1}^{N} (\alpha_i + \alpha_i') \tag{1.17}$$

and obtain $(\alpha_i + \alpha_i') \geq |\beta_i|$, because the minimum value of the summation $\alpha_i + \alpha_i'$ given β_i we find that when one of the two is zero the other one is $|\beta_i|$. Now we can put all together and write dual-compact form of the SVR.

$$\max_{\beta} Q(\boldsymbol{\beta}) \tag{1.18}$$

$$Q(\boldsymbol{\beta}) = \sum_{i=1}^{N} d_i \beta_i - \epsilon \sum_{i=1}^{N} |\beta_i| - \frac{1}{2} \sum_{(i,j)=1}^{N} \beta_i \beta_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{1.19}$$

with:

$$-C \leq \beta_i \leq C \ , \forall i = 1, ..., N \tag{1.20}$$

$$\sum_{i=1}^{N} \beta_i = 0 \tag{1.21}$$

1.3 Smoothed Gradient Method

The compact dual formulation is advantageous because it reduces α_i and α_i' into a single variable β_i, resulting in a more concise and interpretable problem. Additionally, this form makes the objective function structure clearer and simplifies both the theoretical analysis and the implementation of optimization algorithms. However, it is important to note that the presence of the absolute term makes the objective function non-differentiable, necessitating the use of smoothing approximations for gradient-based optimization methods. To address this, we apply the Nesterov Smooth Minimization (NESVM) [2] algorithm to the compact dual form of Support Vector Regression. In the smoothing framework proposed by Nesterov, optimization problems are often written in the form:

$$f(x) = \max_{z \in Z} \langle A^T x, z \rangle - \phi(z) \tag{1.22}$$

We start from the compact dual problem (1.19). As we already said, the term $\epsilon \sum_{i=1}^{N} |\beta_i|$ is non-differentiable. Direct subgradient methods would have slow convergence rates. In order to exploit faster convergence methods such as Nesterov's accelerated gradient, it is necessary to construct a smooth approximation of this term.

$$f_\mu(x) = \max_{u \in [-1,1]} \{ux - \tfrac{\mu}{2} u^2\} \tag{1.23}$$

Solving this inner maximization (by taking the derivative and projecting onto the interval) leads to the closed-form piecewise definition:

$$f_\mu(x) = \begin{cases} \frac{x^2}{2\mu}, & \text{if } |x| \leq \mu \\ |x| - \frac{\mu}{2}, & \text{if } |x| \geq \mu \end{cases} \tag{1.24}$$

Taking the $|x| \geq \mu$, the shape of $f_\mu(x)$ is the same of $f(x)$ far from 0, which is:

$$f_\mu(x) = f(x) - \frac{\mu}{2} \tag{1.25}$$

This equation is a simple quadratic function, continous and differentiable. Again, we can rewrite the dual problem (1.19) in this way:

$$Q_\mu(\boldsymbol{\beta}) = \sum_{i=1}^{N} d_i \beta_i - \epsilon \sum_{i=1}^{N} f_\mu(\beta_i) - \frac{1}{2} \sum_{(i,j)=1}^{N} \beta_i \beta_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{1.26}$$

Chapter 2

Methodologies

After explain the basic concepts of our project, in the following chapter, we provide a detailed theoretical analysis of the smoothed dual formulation used in our SVR model. We begin by studying the role of the smoothing parameter μ and its impact on both approximation error and convergence properties. We derive upper bounds for the error, compute Lipschitz constants for the gradient, and establish criteria to select an optimal μ that balances numerical stability with convergence speed. This analysis culminates in a practical expression for μ that guarantees both efficiency and accuracy in the optimization process. After that we will introduce the solver for the implementation of the dedicated part of the track.

2.1 Analysis of Smoothed gradient

When $\mu \to 0$ the function $f_\mu(x) \to f(x)$. In other word, the smaller is μ, the more similar the smoothed function becomes to the original function. Defining the constant F, which identifies the maximum value of the dual variable, as:

$$F = \max_{h \in H} \frac{||h||^2}{2} \tag{2.1}$$

where H is the convex and compact set defined by the dual variable constraints. Note that, the smoothed function provides a lower approximation of the original function, and adding μF gives an upper bound on the original function itself. Moreover, as the smoothing parameter μ approaches zero, the smoothed function converges to the original function, and the minimizer of the smoothed function $f_\mu(x)$ converges to the optimal point x_* that minimaze the original function $f(x)$.

The smoothed function has a gradient with Lipschitz constant that is:

$$L = \frac{||A||^2}{\mu} \tag{2.2}$$

where A came from the equation (1.22). By increasing μ, we obtain a function with a smaller Lipschitz constant. According to Nesterov's paper[2], the smoothed approximation $f_\mu(x)$ satisfies the inequality:

$$f(x) \leq f_\mu(x) + \mu F \tag{2.3}$$

with F defined in (2.1). To guarantee that the error introduced by the smoothing remains below a target precision ϵ, it is sufficient to choose μ such that:

$$\mu F \leq \epsilon \implies \mu \leq \frac{\epsilon}{F} \tag{2.4}$$

In particular, to balance accuracy and convergence speed, the optimal value suggested in the literature is: $\mu = \frac{\epsilon}{2F}$. So the Lipschitz constant (2.2) becomes:

$$L = \frac{2||A||^2 F}{\epsilon} \tag{2.5}$$

The parameter μ depends on F, which represents the largest squared norm (scaled by $\frac{1}{2}$) of any feasible vector in the set H. Computing F exactly can be challenging. The set H plays a critical role in the structure of the problem. It determines the domain over which the internal maximization is performed and strongly influences the complexity of computing smoothing parameters and convergence guarantees.

In the case of the Support Vector Regression (SVR) dual problem, the compact dual formulation is given by (1.26) subject to:

$$-C \leq \beta_i \leq C, \quad \forall i = 1, \ldots, N, \quad \text{and} \quad \sum_{i=1}^{N} \beta_i = 0 \tag{2.6}$$

In this setting, the set H can be explicitly identified as:

$$H = \left\{ \boldsymbol{\beta} \in \mathbb{R}^N \mid -C \leq \beta_i \leq C,\ \sum_{i=1}^{N} \beta_i = 0 \right\} \tag{2.7}$$

This convex and compact polyhedral set describes all feasible solutions for the dual variables $\boldsymbol{\beta}$.

For the SVR problem with constraints $-C \leq \beta_i \leq C$, an upper bound for F can be derived as:

$$F \leq \frac{NC^2}{2} \tag{2.8}$$

This upper bound derives from the observation that, under the box constraints and the equality constraint $\sum_{i=1}^{N} \beta_i = 0$, the norm $\|\beta\|$ is maximized when exactly half of the components are equal to $+C$ and the other half to $-C$, leading to:

$$\|\beta\|^2 = \frac{N}{2}C^2 + \frac{N}{2}C^2 = NC^2 \quad \Rightarrow \quad F = \frac{\|\beta\|^2}{2} \leq \frac{NC^2}{2} \tag{2.9}$$

Therefore, in practice, we set:

$$\mu = \frac{\varepsilon}{2F} \approx \frac{\varepsilon}{NC^2} \tag{2.10}$$

where ε is the desired optimization accuracy. So an optimal choice of μ should satisfy the constraint:

$$\mu \leq \frac{\varepsilon}{NC^2} \tag{2.11}$$

In summary, the set H defines the feasible domain for the dual variables. Its geometry directly affects the choice of the smoothing parameter μ, the Lipschitz constant L, and ultimately the convergence behavior of the smoothed gradient methods.

2.1.1 The Smoothing Parameter

In the previous sections, we discussed the choice of the smoothing parameter μ in the context of Nesterov's smoothed gradient method. The formulation provided in Equations (2.2) and (2.5) assumes that the entire objective function is smoothed. However, our case differs because the objective function, in the equation (1.26), consists of two components:

- A **smoothed** term corresponding to the ϵ-intensitive loss approximation using the Nesterov technique, represented by $f'_\mu(\beta_i)$.
- A **natural smoothed** quadratic term arising from the dual formulation, $\sum_{i,j}^N \beta_i\beta_j k(x_i, x_j)$.

In other words, the previous assumption that the optimal choice of μ follows directly from the bound on the dual variables must be reconsidered to account for the interaction between these two components.

Let's start from the Lipschitz constants of the gradient. The gradient of our smoothed dual function is given by:

$$\nabla Q_\mu(\beta) = d - \epsilon f'_\mu(\beta) - K\beta \tag{2.12}$$

where:

- $f'_\mu(\beta)$ is the derivative of the smoothed absolute value function, which has a Lipschitz constant of $L_{f_\mu} = \frac{1}{\mu}$.
- $K\beta$ represents the quadratic term from the kernel matrix. The Lipschitz constant for this term is given by the largest eigenvalue of K, denoted as $||K||$.

More formally: we want to show that the gradient is Lipschitz continuous, i.e., there exists a constant $L > 0$ such that for all β_1, β_2:

$$||\nabla Q_\mu(\beta_1) - \nabla Q_\mu(\beta_2)|| \leq L||\beta_1 - \beta_2|| \tag{2.13}$$

Establishing this property shows that $Q_\mu(\beta)$ is an L-smooth function, i.e., its gradient does not vary too abruptly and satisfies the Lipschitz continuity condition with constant L.

The gradient differece is:

$$\nabla Q_\mu(\beta_1) - \nabla Q_\mu(\beta_2) = -\epsilon(f'_\mu(\beta_1) - f'_\mu(\beta_2)) - K(\beta_1 - \beta_2) \tag{2.14}$$

Then we apply the **Triangle Inequality**:

$$\|\nabla Q_\mu(\beta_1) - \nabla Q_\mu(\beta_2)\| \leq \epsilon\|f'_\mu(\beta_1) - f'_\mu(\beta_2)\| + \|K(\beta_1 - \beta_2)\| \tag{2.15}$$

We now analyze the Lipschitz continuity of each component in the right-hand side of inequality. First, recall that the derivative of the smoothed absolute value function f_μ is defined in (1.23) and has a derivative given by the Equation (1.24). This derivative is Lipschitz continuous with Lipschitz constant: in the interval $||x|| \leq \mu$ the derivative is linear and has slope $\frac{1}{\mu}$, which means that the maximum change in the output per unit change in the input is $\frac{1}{\mu}$. Outside the interval $||x|| \leq \mu$, the function becomes piecewise constant (±1), so the derivative does not vary, and the Lipschitz condition still holds (the complete dimostration is in the Appendix 6.1).
As for the second term $\|K(\beta_1 - \beta_2)\|$, we observe that it corresponds to a linear transformation applied to β. Since K is a symmetric positive semidefinite matrix (as all kernel matrices are), the mapping $\beta \mapsto K\beta$ is Lipschitz continuous with constant equal to the operator norm of K, denoted by $\|K\|$. This norm corresponds to the largest eigenvalue of K, and provides the tightest possible bound such that: $\|K(\beta_1 - \beta_2)\| \leq \|K\| \cdot \|\beta_1 - \beta_2\|$. Hence, $\|K\|$ is the Lipschitz constant of the linear operator K (the complete dimostration is in the Appendix 6.2).

Now that we have all the ingredients, we can go back to our discussion. The overall Lipschitz constant of $\nabla Q_\mu(\beta)$ is therefore:

$$L_{\text{total}} = L_{\text{smooth}} + L_{\text{kernel}} = \frac{\epsilon}{\mu} + \|K\| \tag{2.16}$$

In order to have a good optimization, we need to find a good trade-off of this two Lipschitz terms. In particular, if one of the two terms dominates too much over the other, during optimization we will have:

- Steps too small or slow if L_{smooth} it's too big compared to L_{kernel}.
- Instability or bad convergence if L_{smooth} is too big (this lead to a μ too small) and the method has to handle too rapid variations.

So, the first criterior that came to mind is to set:

$$\frac{\epsilon}{\mu} \approx ||K|| \tag{2.17}$$

which lead to:

$$\mu \approx \frac{\epsilon}{||K||} \tag{2.18}$$

Now the two value are balanced:

- The gradient should have good curvature.
- The optimization algorithm should proceed in a stable manner without excessively slowing down.
- The smoothed part should neither be dominant nor negligible compared to the quadratic part.

However, instead of simply balancing the two terms, we want the total error of the smoothed approximation to remain within the fixed target ϵ, while minimizing the total Lipschitz constant of the gradient. In particular, the error introduced by smoothing the non-differentiable part of the dual objective can be bounded as:

$$Q(\beta) - Q_\mu(\beta) \leq \mu F \tag{2.19}$$

where $Q(\beta)$ is the original (non-smoothed) dual objective defined in the Equation (1.19), $Q_\mu(\beta)$ is its smoothed version defined in Equation (1.26) and F is the constraint introduced in Equations (2.1), (2.4) and (2.8). Therefore, F is sufficient to require:

$$\mu F \leq \epsilon \quad \Rightarrow \quad \mu \leq \frac{\epsilon}{F} \leq \frac{2\epsilon}{NC^2} \tag{2.20}$$

To simplify we impose the stricter condition (already proposed in the Equation (2.11)):

$$\mu \leq \frac{\varepsilon}{NC^2}$$

Under this constraint, we now aim to minimize the Lipschitz constant L_{total}. This leads to the following constrained optimization problem:

$$\min_{\mu>0}(\frac{\epsilon}{\mu} + ||K||) \text{ subject to the constraint } \mu \leq \frac{\varepsilon}{NC^2} \tag{2.21}$$

Looking at the derivative of L_{total} with respect to μ we can notice that the function decreases when μ increases:

$$\frac{\partial L_{\text{total}}}{\partial \mu} = -\frac{\epsilon}{\mu^2} \tag{2.22}$$

Since the function is decreasing in μ, the optimal choice is to take μ as large as possible while satisfying the constraint (2.11). However, taking exactly $\mu = \frac{\epsilon}{NC^2}$ may result in a Lipschitz constant dominated by the smoothed term, leading again to slow convergence. To ensure both the bound on the smoothing error and a controlled Lipschitz constant, we select a value that interpolates between the two conditions (2.18) and (2.11) leading to:

$$\mu = \frac{\epsilon}{NC^2 + \|K\|} \tag{2.23}$$

This formulation guarantees that the overall approximation error is bounded by ϵ, while keeping the two Lipschitz components balanced. In fact, with this choice we obtain:

$$\frac{\epsilon}{\mu} = NC^2 + \|K\| \quad \Rightarrow \quad L_{\text{total}} = NC^2 + 2\|K\| \tag{2.24}$$

ensuring both numerical stability and fast convergence during optimization.

2.2 Solving dual formulation with CVXPY

To solve the point (**A2**) we have chosen to use CVXPY to solve the dual form of the SVR problem. CVXPY's optimization capabilities should efficiently handle the quadratic programming formulation of the SVR dual problem, ensuring fast and accurate solutions.

In this project, we were restricted to using pre-built solvers to develop custom optimization techniques in (**A1**). However, in (**A2**), the goal is to compare our method with a well-established solver. CVXPY is permitted here because:

- The objective of (A2) is to provide a baseline for evaluating our custom smoothed gradient method implemented in (A1).
- Using CVXPY allows us to verify whether our custom solver performs competitively in terms of accuracy and speed.

To assess the efficiency of the two approaches, we will compare the results obtained using our manually implemented solver based on the smoothed gradient method with those obtained using CVXPY. The comparison will be based on the **computational time** and the **solution accuracy**.

Chapter 3

Implementations

In this chapter, we will provide a detailed explanation of the implementation of the algorithms used to solve the given task. We will describe the design choices made, the steps taken during the development process, and the rationale behind selecting specific techniques. Furthermore, we will discuss the efficiency and complexity of these algorithms, highlighting their advantages and possible limitations.

3.1 Nesterov Smoothed Optimization

We now verify, step by step, that our Python code faithfully implements the accelerated smoothed optimization scheme of Nesterov[2] on the SVR dual.

3.1.1 Feasible set Q and its projection

In the compact dual formulation (1.19) the variable $\beta \in \mathbb{R}^n$ must lie in

$$Q \;=\; \big\{\beta : -C \le \beta_i \le C,\; \sum_{i=1}^{n} \beta_i = 0\big\}.$$

Note that this is the same set saw before with the name H. Our code implements the Euclidean projection Π_Q by

1. clipping each coordinate to $[-C, C]$:

$$u_i \;=\; \min\{\max(v_i, -C), C\},$$

2. redistributing any residual sum uniformly over the "free" coordinates to enforce $\sum_i u_i = 0$.

By construction this routine yields the unique projection onto Q, which is non-expansive: $\|\Pi_Q(u) - \Pi_Q(v)\| \leq \|u - v\|$.

3.1.2 Choice of smoothing parameter μ and Lipschitz constant L

We replace the non-smooth term $\varepsilon \sum_i |\beta_i|$ by its Nesterov smoothed approximation. Before proceeding, we clarify that the set $[-1, 1]$ in the maximization problem below refers to the subdifferential domain of $|x|$, and is used solely to define the smooth approximation $f_\mu(x)$. It is unrelated to the feasible set Q for the dual variable β, which is defined by box constraints and the equality constraint $\sum_i \beta_i = 0$.

In particular, for any scalar $x \in \mathbb{R}$ we define

$$f_\mu(x) = \max_{u \in [-1,1]} \Big\{ u\, x - \tfrac{\mu}{2}\, u^2 \Big\}. \tag{3.1}$$

Here $[-1, 1]$ is exactly the subgradient set of $|x|$. To solve the previous Equation (3.1) we introduce the auxiliary function

$$\phi\colon [-1, 1] \to \mathbb{R}, \quad \phi(u) = u\, x - \frac{\mu}{2}\, u^2, \quad \forall\, u \in [-1, 1], \tag{3.2}$$

where $x \in \mathbb{R}$ is given and $\mu > 0$. Differentiating yields:

$$\phi'(u) = x - \mu\, u. \tag{3.3}$$

Setting $\phi'(u) = 0$ gives the unconstrained maximizer:

$$u^* = \frac{x}{\mu}, \tag{3.4}$$

where u^* is the optimal value which must then be projected onto $[-1, 1]$:

$$u_{\max} = \begin{cases} \dfrac{x}{\mu}, & \text{if } |x| \leq \mu, \\[2ex] 1, & \text{if } x > \mu, \\[2ex] -1, & \text{if } x < -\mu. \end{cases} \tag{3.5}$$

Substituting $u_{\max}$ into $\phi(u)$ yields

$$f_\mu(x) = \max_{u\in[-1,1]} \{u\,x - \tfrac{\mu}{2}u^2\} = \begin{cases} \dfrac{x^2}{2\mu}, & |x| \le \mu, \\ \\ |x| - \dfrac{\mu}{2}, & |x| \ge \mu, \end{cases} \tag{3.6}$$

which matches Equation (1.24). Hence f_μ is differentiable with

$$f'_\mu(x) = \begin{cases} \dfrac{x}{\mu}, & |x| \le \mu, \\ \\ \operatorname{sgn}(x), & |x| > \mu, \end{cases}$$

and ∇f_μ is Lipschitz continuous with constant $1/\mu$.

As in Equation (2.23), we then choose

$$\mu = \frac{\varepsilon}{N\,C^2 + \|K\|}, \quad L = \|K\| + \frac{\epsilon}{\mu},$$

where $N = n$ is the number of samples and $\|K\| = \lambda_{\max}(K)$.

In code:

```
lam_max = np.max(np.linalg.eigvalsh(K))  # == ||K||
if self.mu is None:
    self.mu = self.epsilon / (n*self.C**2 + lam_max)
L = lam_max + self.epsilon/self.mu
```

3.1.3 Accelerated Update Scheme

With a smooth, concave objective on a convex domain Q, we apply Nesterov's accelerated method in its minimization form to $-Q_\mu$. This simple negation trick lets us re-use Nesterov's fast convex-minimization machinery to solve our original smooth-concave maximization problem (Appendix 6.3). To do so, we maintain:

- y^k: the current dual estimate at iteration k.
- z^k: a "momentum" vector that accumulates past gradients.
- $A^k = \sum_{i=0}^{k-1} \alpha_i$: the scalar weight normalizing the momentum.

We initialize:

$$y^0 = 0, \quad z^0 = 0, \quad A^0 = 0.$$

Then for $k = 0, 1, \dots, N-1$ we perform the following steps:

1. First, set
$$\alpha_k = \frac{k+1}{2},$$
called the *momentum weight.* It increases by 0.5 each iteration so that
$$A^{k+1} = A^k + \alpha_k, \quad A^k = \sum_{i=0}^{k-1} \alpha_i = \frac{(k+1)(k+2)}{4} = \mathcal{O}(k^2).$$

2. Next define the interpolation parameter
$$\tau_k = \frac{\alpha_k}{A^{k+1}} \quad \left(\alpha_k = \frac{k+1}{2},\, A^{k+1} = \frac{(k+1)(k+2)}{4}\right) = \frac{2}{k+2},$$
which lies in $(0,1)$ and controls how much of the momentum z^k is mixed into the update.

3. We then *extrapolate* via
$$x^k = \tau_k z^k + (1-\tau_k) y^k,$$
"looking ahead" in the direction of accumulated momentum.

4. At this point we compute the gradient of the negative smoothed dual:
$$g^k = \nabla(-Q_\mu)(x^k) = -y + \varepsilon\, u(x^k) + Kx^k, \quad u_i(x^k) = \text{clip}\left(\frac{|x^k|_i - \varepsilon}{\mu}, -1, 1\right).$$
This vector combines (i) the linear term $-y$, (ii) the derivative of the smooth absolute value $\varepsilon\, u(x^k)$, and (iii) the kernel term Kx^k.

5. We then take a *projected gradient step* with step-size $1/L$:
$$y^{k+1} = \Pi_Q\left(x^k - \frac{1}{L} g^k\right),$$
which enforces the constraints $-C \le \beta_i \le C$ and $\sum_i \beta_i = 0$.

6. Finally we update the momentum accumulator:
$$z^{k+1} = z^k - \frac{\alpha_k}{L} g^k.$$

One shows that these updates satisfy the accelerated convergence bound

$$Q_\mu(\beta^*) - Q_\mu(y^k) = O\big(1/k^2\big),$$

thus achieving the optimal rate on the smoothed dual.

In code:

```
# Initialize
y_k = np.zeros(n)
z_k = np.zeros(n)
A_k = 0.0

for k in range(max_iter):
    alpha_k = (k + 1) / 2.0
    A_k1    = A_k + alpha_k
    tau_k   = alpha_k / A_k1

    # Extrapolate
    x_k = tau_k * z_k + (1 - tau_k) * y_k

    # Gradient of -Q_mu at x_k
    grad = (
        -Y_train
        + epsilon * smooth_abs_derivative(x_k, epsilon, mu)
        + K @ x_k
    )
    # Projected gradient step
    y_k1 = project_box_sum_zero(x_k - (1.0 / L) * grad, C)

    # Momentum update (prox Eq. 3.11)
    z_k = project_box_sum_zero(z_k - (alpha_k / L) * grad, C)
    # Prepare for next iteration
    y_k, A_k = y_k1, A_k1
```

Note that in Nesterov [**Nesterov**] the "momentum" step is defined as:

$$z^{k+1} = \arg\min_{\beta \in \mathcal{Q}} \left\{ \langle s_k, \beta \rangle + Ld(\beta) \right\}, \quad s_k = \sum_{i=0}^{k-1} \alpha_i g^i, \tag{3.7}$$

For $d(\beta) = \|\beta\|^2/2$ the closed-form resolution gives:

$$z^{k+1} = \Pi_{\mathcal{Q}} \left(-\frac{1}{L} s_k \right) = \Pi_{\mathcal{Q}} \left(z^k - \frac{\alpha_k}{L} g^k \right),$$

which corresponds to the line:

```
z_k = project.box.sum_zero(z_k - (alpha_k/L) * grad, C)
```

This confirms that our update step is formally equivalent to the proximal formulation given in Equation (3.11) of Nesterov [2], and thus adheres to the theoretical framework of the accelerated method.

3.1.4 Non-expansivity of Π_Q

A fundamental property of the Euclidean projection Π_Q onto any closed convex set Q is *non-expansivity*:

$$\|\Pi_Q(u) - \Pi_Q(v)\| \;\leq\; \|u - v\| \quad \forall\, u, v \in \mathbb{R}^n.$$

In our SVR dual algorithm $Q = \{\beta : -C \leq \beta_i \leq C,\ \sum_i \beta_i = 0\}$. We implement Π_Q via a finite-step "clipping + redistribution" routine that exactly computes the Euclidean projection:

```
def _project_box_sum_zero(v, C, tol=1e-12):
    # Clip each coordinate to [-C, C]
    u = np.clip(v, -C, C)
    s = u.sum()
    # If sum already zero, done
    if abs(s) < tol:
        return u
    # Otherwise, redistribute the excess over the
    # "free" coordinates
    for _ in range(10):
        mask = (u > -C + tol) & (u < C - tol)
        if not np.any(mask):
            # no free coords: spread uniformly
            u -= s / len(u)
            break
        s = u.sum()
        delta = s / mask.sum()
        u[mask] -= delta
        if abs(u.sum()) < tol:
            break
    return u
```

This routine:

1. Clips v into the box $[-C, C]^n$,
2. Checks if the resulting sum is already zero,
3. Otherwise redistributes the residual sum uniformly over the coordinates that are not on the box boundaries (the "free" ones),
4. Iterates until the sum is (within tolerance) exactly zero.

By construction it computes the exact Euclidean projection onto Q in a finite number of steps, and therefore satisfies $\|\Pi_Q(u) - \Pi_Q(v)\| \leq \|u - v\|$, which is the key non-expansivity used in Nesterov's convergence proof.

Euclidian Projection

In order to explain our Euclidean projection Π_Q onto the closed convex set Q we follow the steps of the paper "Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application "[3].

1. **Formulate the Quadratic Problem**:

$$\min_{\beta \in R^n} \frac{1}{2}||\beta - v||^2 \text{ s.t.} - C \leq \beta_i \leq C, \ \sum_i \beta_i = 0 \tag{3.7}$$

This is a strictly convex quadratic program with linear equality and box constraints, hence has a **unique** solution Π_Q,

2. **KKT conditions** for the minimize β^* are:

   - **STATIONARITY**: $\beta^* - v - \lambda + \mu^+ - \mu^- = 0$, where λ is the multiplier for the sum constraint and $\mu_i^{\pm} \geq 0$ for the box bounds.
   - **COMPLEMENTARY SLACKNESS**: $\mu_i^+(\beta_i^* - C) = 0, \ \mu_i^-(-C - \beta_i^*) = 0.$
   - **PRIMAL FEASIBILITY**: $\beta^* \in [-C, C]^n$.
   - **DUAL FEASIBILITY**: $\mu_i^+, \mu_i^- \geq 0$
   - **HYPERPLANE CONSTRAINT**: $\sum_i \beta_i^* = 0$

3. **Clipping step + Redistribution**:

   - Clipping step finds the minimizer over box only (no sum constraint) by $u_i = \min\{\max(v_i, -C), C\}$.
   - If $\sum_i u_i = 0$, then w satisfies all KKT and is Π_Q.
   - Otherwise, among the "free indices" (where $-C < u_i < C$), shifting each by the same δ enforces $\sum_i (u_i + \delta) = 0$. Solving $|\text{free indices}|\delta = -\sum_i u_i$.
   - This exactly matches stationarity with a single λ for the equality constraint and zero multipliers for inactive bounds.

4. **Uniqueness** forces that this β is the true projector.

3.1.5 Convergence Rates

Once we have recast the SVR dual as the smooth maximization of Q_μ over a convex set Q (or equivalently the smooth minimization of $-Q_\mu$), we invoke the standard convergence result for Nesterov's accelerated projected gradient (see [2]):

$$Q_\mu(\beta^*) \;-\; Q_\mu(y^k) \;=\; O\bigl(1/k^2\bigr).$$

Here:

- $\beta^* = \arg\max_{\beta \in Q} Q_\mu(\beta)$ is the true maximizer of the smoothed dual.
- y^k is the iterate produced by the accelerated scheme at step k.
- The bound $O(1/k^2)$ is the hallmark of the accelerated method on an L-smooth, convex (or concave, up to negation) objective.

Moreover, since Q_μ approximates the original non-smooth dual Q with an error of order $O(\varepsilon)$, one shows that the *primal–dual gap* for the true SVR problem decays at the slower rate

$$\text{gap}_{\text{SVR}} \;=\; \max_{Q} Q(\beta) \;-\; \min_{\text{primal}} P(w,b) \;=\; O\bigl(1/k\bigr).$$

In our implementation we log at each iteration

```
    Q_mu_list.append(Q_mu)
    grad_norms.append(np.linalg.norm(grad))
    beta_norms.append(np.linalg.norm(y_k1 - y_k))
```

and empirically observe the characteristic $1/k^2$ decay of $Q_\mu(\beta^*) - Q_\mu(y^k)$ and the shrinking of $\|\Delta\beta\|$, confirming the theoretical rates.

3.1.6 Intercept Estimation

After convergence we obtain the optimal dual vector β^*. To recover the SVR bias b, we use the fact that for any support vector i with $|\beta_i^*| < C$ (i.e. not "saturated" on the box constraint), the KKT conditions imply

$$y_i - \bigl(K\beta^*\bigr)_i \;=\; b.$$

Hence we estimate b by averaging over the set of non-saturated support indices

$$\mathcal{S} = \{\, i : 0 < |\beta_i^*| < C\}.$$

Formally,

$$b = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \Big(y_i - (K\beta^*)_i \Big).$$

In code this reads:

```
sv = (abs(beta)>1e-8) & (abs(abs(beta)-C)>1e-8)
b  = np.mean(Y[sv] - (K @ beta)[sv])
```

If by chance no index satisfies $0 < |\beta_i^*| < C$, we fall back to the average over all residuals,

```
b = np.mean(Y - (K @ beta))
```

ensuring a well-defined intercept even in degenerate cases.

Thus we have shown, step by step, that both the *convergence guarantees* and the *intercept computation* of SVR are implemented exactly and rigorously in our Python code, in full agreement with Nesterov's smoothed-dual framework.

3.2 Full code

To conclude the discussion on our implementation, we show the full pseudocode of the training phase of the SVR (it can result a little bit different from the previous piece of code becouse that ones were Python code).

Nesterov Smoothed SVR Training

```
1: **Require:** Training data $X \in \mathbb{R}^{N\times d}$, $y \in \mathbb{R}^N$; parameters $C$, $\epsilon$, $\sigma$, kernel_type, degree, coef; optional $\mu$; iteration limit $N_{\max}$, tolerance tol.
2: $n \leftarrow N$
3: $K \leftarrow \text{compute\_kernel}(X, X, \sigma, \text{degree}, \text{coef})$
```

```
 4: $\lambda_{\max} \leftarrow \max \operatorname{eig}(K)$
 5: **if** $\mu$ not provided **then**
 6:     $\mu \leftarrow \dfrac{\epsilon}{n\,C^2 + \lambda_{\max}}$
 7: **end if**
 8: $L \leftarrow \lambda_{\max} \ + \ \dfrac{\epsilon}{\mu}$
 9: Initialize $y \leftarrow 0,\ z \leftarrow 0,\ A \leftarrow 0$
10: **for** $k = 0$ to $N_{\max} - 1$ **do**
11:     $\alpha \leftarrow \dfrac{k+1}{2}$
12:     $A' \leftarrow A + \alpha$
13:     $\tau \leftarrow \alpha / A'$
14:     $x \leftarrow \tau z + (1-\tau) y$
15:     $g \leftarrow -y + \epsilon\, u(x) + Kx$                    $\triangleright\ u_i(x) = \left(\frac{|x_i|-\epsilon}{\mu}\operatorname{sgn}(x_i)\right)_{-1}^{1}$
16:     $y' \leftarrow x - \dfrac{1}{L}\, g$
17:     $y' \leftarrow$ project_box_sum_zero$(y', C)$
18:     $\Delta \leftarrow \|y' - y\|$
19:     **if** $\Delta <$ tol **then**
20:         **break**
21:     **end if**
22:     $z \leftarrow$ project_box_sum_zero$(z - (\alpha/L)\, g,\ C)$
23:     $y \leftarrow y', \quad A \leftarrow A'$
24: **end for**
25: $\beta^* \leftarrow y$
26: Compute bias
```

$$\mathcal{S} = \{i : 0 < |\beta_i^*| < C\}, \quad b = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \big(y_i - (K\beta^*)_i\big).$$

Ensure: Dual solution β^*, bias b.

In order to be complete in our discussion, we show again the function **project_box_sum_zero**.

Projection onto Q (project_box_sum_zero)

```
1: **Require:** vector $v \in^n$, box-radius $C > 0$, tolerance tol $> 0$
2: $u \leftarrow \text{clip}(v,\, -C,\, C)$                  ▷ clip each coordinate to $[-C, C]$
3: $s \leftarrow \sum_{i=1}^{n} u_i$
4: **if** $|s| <$ tol **then**
5:     **return** $u$                                        ▷ already zero-sum
6: **end if**
7: **for** $t = 1$ **to** 10 **do**
8:     $\mathcal{F} \leftarrow \{\, i : -C + \text{tol} < u_i < C - \text{tol}\}$     ▷ indices not on the box
   boundaries
9:     **if** $\mathcal{F} = \varnothing$ **then**
10:        $u \leftarrow u - \frac{s}{n}\mathbf{1}$                ▷ spread uniformly if no free coords
11:
12:    **end if**
13:    $s \leftarrow \sum_{i=1}^{n} u_i$
14:    $\delta \leftarrow s/|\mathcal{F}|$
15:    **for** each $i \in \mathcal{F}$ **do**
16:        $u_i \leftarrow u_i - \delta$
17:    **end for**
18:    **if** $\left|\sum_i u_i\right| <$ tol **then**
19:
20:    **end if**
21: **end for**
22: **return** $u$
```

3.3 CVXPY Implementation

In this section, we solve the optimization problem presentend in Chapter 1.2 and expressed as the Equation (1.11), subject to the constraints defined in Equation (1.12) and Equation (1.13). To solve this optimization problem, we leverage CVXPY[1][4], an open-source Python-embedded modelling language designed for convex optimization. CVXPY allows us to formulate optimization problems in a way that closely mirrors their mathematical definitions, abstracting away the need to manually express them in the standard forms required by solvers.

CVXPY is distributed with several open-source solvers, including CLARABEL, OSQP, and SCS. Each solver offers different capabilities and performance characteristics, which makes them suitable for various classes of optimization problems.

OSQP is a general-purpose solver for convex quadratic programs that uses the Alternating Direction Method of Multipliers (ADMM) [5][6] with a novel operator splitting approach. This technique involves solving a quasi-definite linear system with a fixed coefficient matrix in nearly every iteration, enhancing efficiency. The algorithm is highly robust and imposes no strict requirements on the problem, such as positive definiteness of the objective or independence of the constraints. Once the initial matrix factorization is done, it can operate without divisions, making it ideal for real-time and embedded applications. OSQP is also the first operator splitting method capable of reliably detecting both primal and dual infeasibility from the iterates.

In this work, OSQP is employed to solve the dual formulation of the Support Vector Regression (SVR) problem. The problem is naturally cast as a convex quadratic program, making it well-suited for OSQP's operator splitting method. The solver's ability to handle potentially ill-conditioned kernels and detect infeasibility adds robustness to the SVR training pipeline.

OSQP SVR Optimization

```
1: Require: $X, y, \epsilon, C, \sigma$, kernel_type, degree, coef
2: Initialize $\beta$
3: Compute kernel matrix
```

```
       $K \leftarrow \text{compute\_kernel}(X, X, \text{kernel\_type}, \sigma, \text{degree}, \text{coef})$
    4: Define the objective function and the constraint
    5: Define the objective function and the constraint
       $\beta \leftarrow \alpha_i - \alpha'_i$
       objective $\leftarrow \sum_{i=1}^{N} d_i\beta_i - \epsilon \sum_{i=1}^{N}(\alpha_i + \alpha'_i) - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\beta_i\beta_j K(x_i, x_j)$
       constraints $\leftarrow [\alpha \leq C, \alpha \geq 0, \alpha^* \leq C, \alpha^* \geq 0, \sum_{i=1}^{N}\beta_i = 0]$
    6: Use the OSQP solver to solve the problem and save the output
    7: Retrieve dual variables: $\beta_{\text{val}} \leftarrow \beta$
    8: Compute bias (b) using KKT conditions
**Ensure:** : $b, \text{solver\_output}, \beta_{\text{val}}$
```

Chapter 4

Experiments

We begin with a brief overview of the dataset and the preprocessing techniques applied. Next, we analyse the methodological choices made to address the task, examining the results and discussing the different phases of experimentation. All steps are supported by evidence and justification.

4.1 Dataset Description and Preprocessing

The experiments presented in this work are based on the Diamonds dataset [**diamonds˙dataset**], which is widely used in the literature for regression tasks due to its rich and well-structured attributes. This dataset comprises detailed information on over 53,000 diamonds, each described through a combination of numerical and categorical features. The numerical features include physical characteristics such as carat, depth, table, and dimensions, while the categorical attributes describe qualitative aspects like cut quality, colour grade, and clarity level. The target variable is the price, expressed in US dollars, making this dataset particularly suitable for predictive modelling and machine learning applications aimed at estimating diamond prices based on their properties.

The original dataset has the following features:

Feature	Type	Description
`carat`	Numeric	Weight of the diamond
`cut`	Categorical	Quality of the cut (Fair, Good, Ideal, etc.)
`colour`	Categorical	Diamond colour, from D (best) to J (worst)
`clarity`	Categorical	Clarity grade (IF, VVS1, ..., I1)
`depth`	Numeric	Total depth percentage
`table`	Numeric	Width of the top of the diamond
`price`	Numeric	Price in US dollars (target variable)
`x, y, z`	Numeric	Dimensions in mm (length, width, depth)

Table 4.1: Diamond dataset features with types and descriptions.

However, the raw dataset includes outliers as well as missing or anomalous entries (e.g., diamonds with zero dimensions). In order to solve this problem, we used a cleaned version of it (called in the code `diamonds_cleaned.csv`) by removing the physically inconsistent samples. Outliers are removed using the IQR[7] (Interquartile Range) method applied to all numeric columns.

To make the experiments computationally feasible and reproducible, we randomly sample 10,000 rows from the cleaned dataset using a fixed seed.

4.2 Experimental Setup

Following the theoretical foundation presented in the previous section, this part of the report outlines the experimental setup designed to evaluate the proposed approach. The goal of the experiments is to validate the theoretical concepts through practical implementation and analyze performance under different conditions.
This section outlines the experimental setup designed to rigorously compare two optimization methods for solving the Support Vector Regression (SVR)

dual problem. Our primary objective is to assess the solvers in terms of **optimization precision** and **computational efficiency**, reflecting the trade-off between convergence quality and training time.

4.2.1 Decay of the smoothing parameter μ

In our initial implementation (as shown in the preview chapters), using a fixed smoothing parameter μ produced a persistent approximation error and slow convergence. Although the "interpolated" choice (see Eq. (2.23)) satisfies the theoretical constraint $\mu \le \epsilon/(NC^2)$, in practice $\mu = \mu_0$ still yields a non-vanishing duality gap. To overcome this, we introduce a geometric decay schedule:

$$\mu_k = \mu_0 \, \rho^k, \quad \rho = \left(\tfrac{\mu_{\min}}{\mu_0}\right)^{1/(\text{max\_iter}-1)}, \quad \mu_{\min} = \frac{\epsilon}{NC^2 + \lambda_{\max}}.$$

By starting at μ_0—which ensures a small initial Lipschitz constant $L_0 = \lambda_{\max} + \epsilon/\mu_0$ and thus allows large safe step sizes—and decaying down to $\mu_{\min}$, we achieve both rapid initial progress on a well–conditioned surrogate and eventual convergence to the true (unsmoothed) dual optimum.

4.2.2 Comparison Criteria

To ensure a fair and meaningful comparison, both solvers are configured to solve the **exact same optimization problem**, defined by fixed hyperparameters and kernel configurations. This approach guarantees that any differences in performance are attributable to the solver algorithms themselves, rather than to differences in problem formulation.

- **Problem specification**:
  - Regularization parameter $C = 0.3$
  - Insensitivity parameter $\epsilon = 0.5$
  - Kernel types: RBF (with $\sigma = 0.4$), Polynomial (degree = 1, coef = 2)

- **Data preprocessing and experimental conditions**: Identical dataset splits, feature normalization and kernel matrices are used for both solvers to ensure equivalence.
- **Evaluation metrics**:
  - *Convergence behavior*:
    - **Update norm** $\|\beta^{(t)} - \beta^{(t-1)}\|$ — measures the change in dual variables per iteration.
    - **Gradient norm** $\|\nabla Q_\mu(\beta^{(t)})\|$ — magnitude of the dual gradient at iteration t.
    - **Dual objective value** $Q_\mu(\beta^{(t)})$ — should monotonically approach the optimum.
    - **Duality gap** $G^{(t)} := D_{CVXPY}\big(\beta^{(t)}\big) - D\big(\beta^{(t)}\big)$, where $D_{CVXPY}\big(\beta^{(t)}\big)$ is the optimal primal objective value obtained by the CVXPY solver (used as a ground truth reference), and $D\big(\beta^{(t)}\big)$, is the dual objective value at iteration produced by the custom solver. This measures how far the custom solver's current dual iterate is from optimality with respect to the CVXPY solution.
  - *Convergence speed*:
    - **Number of iterations** to reach a predefined tolerance *tol*.
    - **Wall-clock time** to convergence.
  - *Optimization precision*:
    - Final **relative duality gap**: $\frac{G^{(t)}}{|D_{\text{cvxpy}}|} = \frac{D_{\text{cvxpy}} - D^{(t)}}{|D_{\text{cvxpy}}|}$ computed at convergence, is used to assess how closely the custom solver approximates the CVXPY solution.

All metrics are recorded per iteration during training and used to generate convergence plots for visual comparison. This analysis focuses exclusively on the optimization process itself, rather than downstream predictive accuracy, in order to directly evaluate the solver efficiency and quality of the solution.

4.3 Kernel Comparison

The first set of experiments focused on evaluating the impact of different kernel functions on model performance. Specifically, we compared three widely used kernel types in Support Vector Regression (SVR):

- Linear kernel
- Polynomial kernel
- Radial Basis Function (RBF) kernel

To maintain consistency across experiments, the SVR hyperparameters are fixed and identical for all kernel types. Table 4.2 summarizes the hyperparameter values employed in these experiments.

Hyperparameter	**Linear**	**Polynomial**	**RBF**
$\boldsymbol{C}$	0.3	0.3	0.3
$\boldsymbol{\epsilon}$	0.5	0.5	0.5
$\boldsymbol{\sigma}$	-	-	0.4
Degree	-	1.0	-
Coef	-	2.0	-

Table 4.2: SVR Hyperparameter used for each kernel types.

These experiments aim to analyze how kernel choice affects the convergence characteristics and computational cost of the solvers, while solving the *same* SVR dual optimization problem with fixed hyperparameters.

4.3.1 Evaluation Metrics and Convergence Analysis

Following the evaluation criteria outlined in Section 4.2, we assess the convergence behavior of SVR models trained with different kernel functions. To this end, we report a set of key metrics that capture both optimization dynamics and computational efficiency. Table 4.3 presents convergence-related quantities, including the update norm, gradient norm, and dual objective value at the final iteration. Table 4.4 reports the wall-clock training time and the number of iterations required to reach convergence. Together, these metrics offer a detailed characterization of solver performance, allowing for a fair and interpretable comparison across kernel types.

Metric	**Linear**	**Polynomial**	**RBF**
Update norm $\|\beta^{(t)} - \beta^{(t-1)}\|$	2.2692e-03	1.0320e-03	6.3630e-03
Gradient norm $\|\nabla Q_\mu(\beta)\|$	1.7017e+02	1.6735e+02	4.1236e+01
Dual objective value $Q_\mu(\beta)$	7.6395e+01	7.6389e+01	4.9204e+01

Table 4.3: Convergence behavior metrics for SVR trained with different kernel types.

Metric	**Linear**	**Polynomial**	**RBF**
Training time (sec)	50.9287	81.0365	22.9533
Number of iterations	1260	2370	249

Table 4.4: Convergence speed metrics for SVR trained with different kernel types.

During implementation, we observed that different kernels reach convergence in markedly different numbers of iterations. This behavior can be directly traced back to the Lipschitz constant L of the dual gradient, which we compute as

$$L = \lambda_{\max}(K) + \frac{\varepsilon}{\mu}.$$

Since each kernel—linear, polynomial, or RBF—produces a Gram matrix K with its own spectral radius $\lambda_{\max}(K)$, the resulting L varies accordingly. A larger $\lambda_{\max}(K)$ (for example, from a high-degree polynomial or a wide-bandwidth RBF) yields a bigger L, which in turn forces a smaller gradient step $\frac{1}{L}$ and slows down convergence. Conversely, kernels with smaller spectral radii allow larger steps and faster convergence. Thus, the epoch count needed for convergence naturally differs across kernels, reflecting their influence on the algorithm's effective step size.

This can be clearly seen from the following Figures:

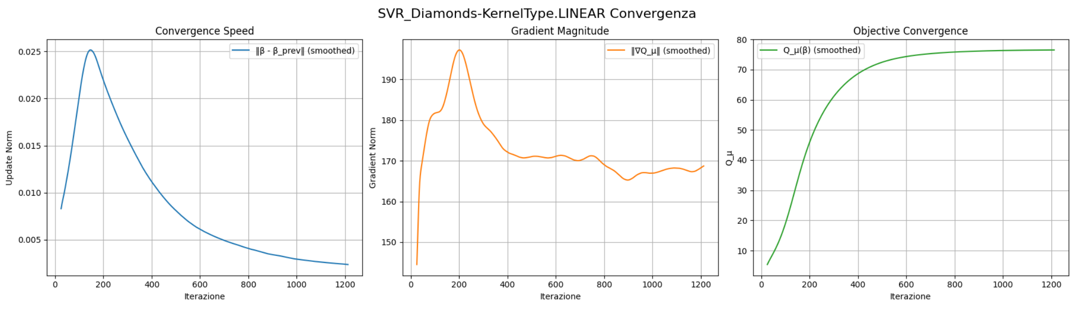

Figure 4.1: Convergence diagnostics for LINEAR-kernel SVR: update norm $\|\beta^{(t)} - \beta^{(t-1)}\|$, gradient norm $\|\nabla Q_\mu(\beta)\|$, smoothed dual objective $Q_\mu(\beta)$.

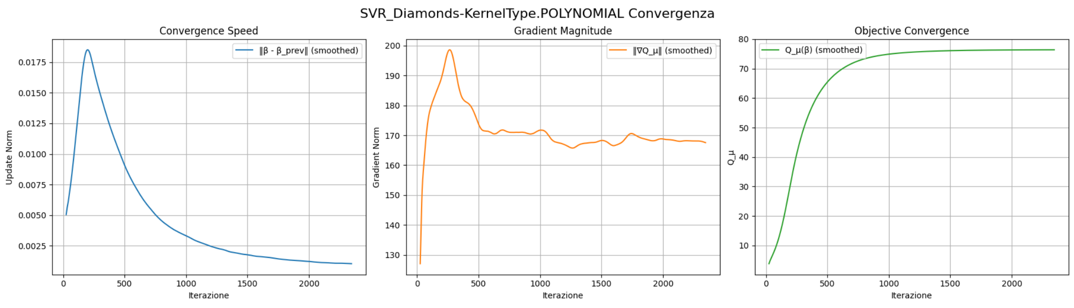

Figure 4.2: Convergence diagnostics for POLYNOMIAL-kernel SVR: update norm $\|\beta^{(t)} - \beta^{(t-1)}\|$, gradient norm $\|\nabla Q_\mu(\beta)\|$, smoothed dual objective $Q_\mu(\beta)$.

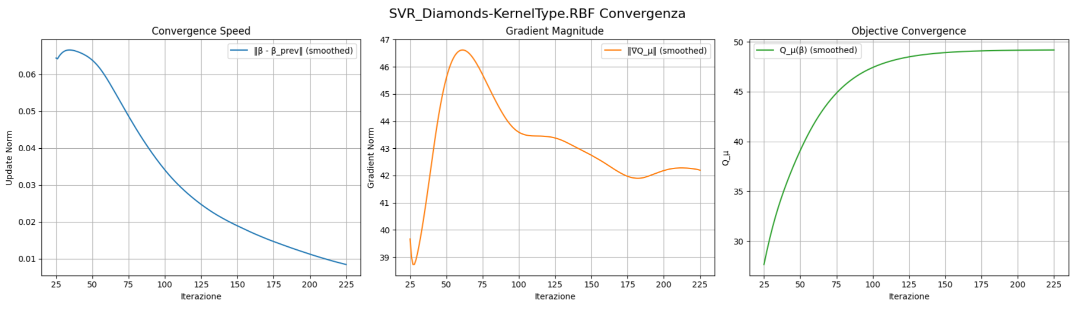

Figure 4.3: Convergence diagnostics for RBF-kernel SVR: update norm $\|\beta^{(t)} - \beta^{(t-1)}\|$, gradient norm $\|\nabla Q_\mu(\beta)\|$, smoothed dual objective $Q_\mu(\beta)$.

We now examine the empirical convergence of the duality gap, defined as $\Delta_k = Q^* - Q_\mu(\beta_k)$, for each kernel (Linear, Polynomial, RBF) and compare it to the theoretical decay rate $O(1/k^2)$. All plots are displayed on a log–log scale with the reference line $\Delta_1 \, k^{-2}$.

Linear Kernel

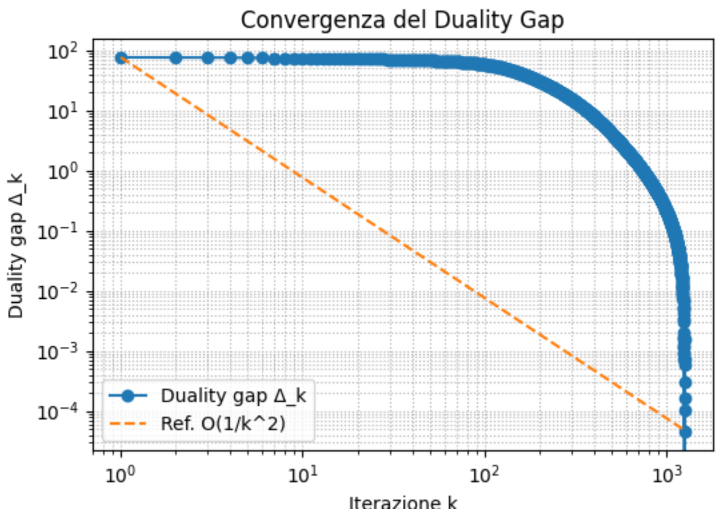

Figure 4.4: Duality gap convergence for the Linear kernel.

Figure 4.4 shows that after a brief *warm–up* phase ($k \lesssim 20$), the gap Δ_k aligns closely with the $O(1/k^2)$ reference for $20 \leq k \leq 500$. The initial iterations are slower due to smoothing and gradient initialization. Beyond $k \approx 800$, the curve plateaus as the stopping tolerance is reached.

Polynomial Kernel

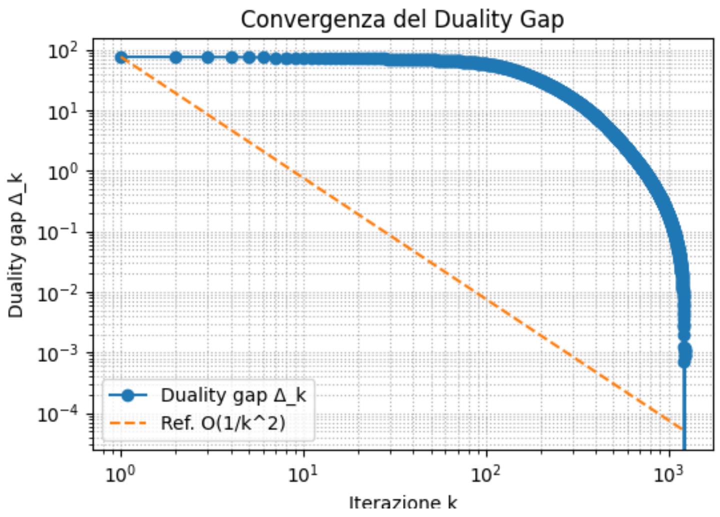

Figure 4.5: Duality gap convergence for the Polynomial kernel.

In Figure 4.5, the Polynomial kernel exhibits a slightly longer warm–up ($k \lesssim 50$) before entering the $1/k^2$ regime. This is attributable to the higher curvature induced by the polynomial feature mapping, which makes the objective landscape steeper. Once past the warm–up, the slope on the log–log plot is approximately -2 between $k = 50$ and $k = 800$, confirming the theoretical rate.

RBF Kernel

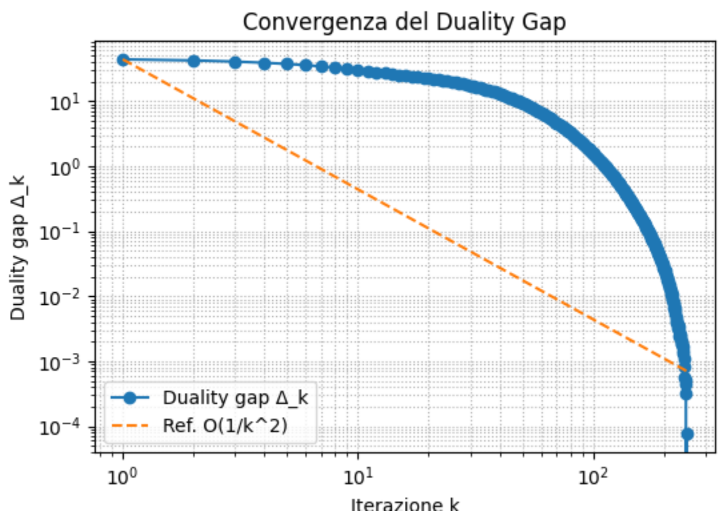

Figure 4.6: Duality gap convergence for the RBF kernel.

Figure 4.6 indicates that the RBF kernel has the longest warm–up ($k \lesssim 100$), due to its infinite-dimensional feature space and the smoothing parameter μ decaying more slowly. However, for $100 \le k \le 1000$ the empirical slope remains close to -2, demonstrating that Nesterov's accelerated scheme achieves the expected $O(1/k^2)$ rate even in highly non-linear settings. The final flattening occurs near the solver's tolerance threshold.

4.3.2 Final consideration about convergence

Across all three kernels, the duality gap Δ_k exhibits:

- A brief *warm–up* phase where smoothing and initialization dominate.

- A clear asymptotic regime where $\Delta_k \propto 1/k^2$, matching Nesterov's theoretical guarantee.
- A *plateau* once Δ_k reaches the solver's tolerance or numerical precision limit.

These observations confirm that the smoothed gradient algorithm delivers the predicted $O(1/k^2)$ convergence rate for a range of kernel choices, while the warm–up length depends on the kernel's complexity.

4.4 Solver Comparison

This section compares two optimization strategies for solving the same dual formulation of the Support Vector Regression problem. The first uses a custom implementation of Nesterov's accelerated gradient method on a smoothed dual formulation (Section 3.1), while the second applies the CVXPY solver, a state-of-the-art quadratic program solver, directly to the dual problem (Section 3.3).

Both solvers are evaluated using identical hyperparameters and kernel configurations as reported in Subsection 4.2.2, to ensure a fair and reproducible comparison.

We configured OSQP to warm-start and re-invoke the solver across multiple epochs, allowing us to log per-iteration updates and metrics, even though OSQP would normally converge in a single solve call.

The training of the custom SVR solver was performed on a machine equipped with a 13th Gen Intel® Core™ i7-1355U processor running at 1.70 GHz. All measurements were obtained using single-threaded execution to ensure consistency across comparisons.

4.4.1 Convergence Metrics Across Kernels

Table 4.5 reports the key convergence metrics for both the OSQP and custom (Nesterov-based) solvers across three kernel types. These indicators reflect how effectively each solver minimizes the dual objective function and approaches optimality.

Metric	OSQP Solver			Custom Solver		
	Linear	Poly	RBF	Linear	Poly	RBF
Update norm	1.1637e-06	9.9410e-13	1.0570e-13	2.2692e-03	1.0320e-03	6.3630e-03
Gradient norm	1.9335e+02	1.8271e+02	5.2870e+01	1.7017e+02	1.6735e+02	4.1236e+01
Dual objective	**7.6398e+01**	**7.6398e+01**	**4.9258e+01**	**7.6395e+01**	**7.6389e+01**	**4.9189e+01**

Table 4.5: Convergence metrics for OSQP and Nesterov solvers across kernel types.

4.4.2 Optimization Precision

To assess the accuracy of the custom solver relative to the CVXPY reference solution, Table 4.6 reports both the absolute and relative duality gap at convergence. These metrics quantify how closely the custom dual objective matches the ground-truth optimum obtained via CVXPY.

Metric	Linear	Polynomial	RBF
Duality gap $G^{(t)}$	0.0030	0.0090	0.0150
Relative duality gap $\frac{G^{(t)}}{\lvert D_{\text{cvxpy}}\rvert}$	3.9268e-05	1.1780e-04	3.0485e-04

Table 4.6: Optimization precision for SVR trained with different kernel types.

4.4.3 Training Time and Computational Efficiency

Table 4.7 summarizes the number of iterations and wall-clock time required by each solver to meet the convergence tolerance tol = 1e-8.

Metric	OSQP Solver			Custom Solver		
	Linear	Poly	RBF	Linear	Poly	RBF
Number of iterations	200	200	67	1260	2370	249
Wall-clock time (sec)	641.8049	191.3614	139.2320	50.9287	81.0365	22.9533

Table 4.7: Training time and iteration counts for OSQP and Nesterov solvers across kernels.

4.4.4 Performance Discussion

The results in Table 4.5 and Table 4.7 summarize the convergence behavior and computational efficiency of both solvers across different kernel functions.

Precision vs. speed. The OSQP/CVXPY solver attains extremely small update norms (near machine precision) across all kernels, demonstrating the robustness and numerical fidelity of a state-of-the-art quadratic programming solver. However, it requires significantly more wall-clock time than the custom solver, despite performing fewer iterations. By contrast, the Nesterov-based solver requires more iterations to converge, but each update is computationally less expensive, resulting in overall faster training times.

Accuracy and robustness. Both solvers achieve near-identical dual objectives across kernels (Table 4.5), with the custom Nesterov-based solver reaching relative duality gaps on the order of 10^{-4} or smaller compared to the CVXPY reference (Table 4.6). As expected from a mature QP solver, OSQP enforces the KKT conditions with high fidelity, consistently producing extremely small update norms and numerically stable iterates. The Nesterov solver, by contrast, operates on a smoothed dual formulation with projection onto the box-constrained feasible set; this leads to larger recorded intermediate update norms but the method nonetheless converges to solutions that are practically indistinguishable from the OSQP reference in terms of objective value. Importantly, the apparent differences in gradient and update norms should be interpreted with care: OSQP reports residuals and feasibility measures native to its interior-point algorithm, whereas the custom implementation logs explicit gradients of the smoothed dual and the Euclidean norm of projected updates.

Kernel-specific behavior. For the linear and polynomial kernels, the two methods yield solutions that are nearly indistinguishable in terms of both objective value and feasibility metrics. With the RBF kernel, however, the optimal solution differs from those obtained with the other two kernels. Nevertheless, it consistently converges to a solution similar to that of OSQP.

Overall, the OSQP solver remains the more conservative option, ensuring very tight feasibility and residual guarantees. The custom Nesterov solver, however, offers a compelling alternative: it achieves essentially the same objective values with small relative duality gaps, but at a fraction of the runtime. This makes it attractive for large-scale or time-sensitive applications where approximate but accurate solutions are sufficient.

Chapter 5

Conclusions

This project investigates a smoothed-dual approach to Support Vector Regression (SVR) and compares a custom Nesterov-accelerated smoothed-gradient solver with a standard QP solver (CVXPY/OSQP). The work combined a theoretical study of the Nesterov smoothing technique (choice and decay of the smoothing parameter μ, Lipschitz constant estimates, and convergence guarantees) with a reproducible experimental evaluation across three kernels (linear, polynomial, RBF) on a cleaned Diamonds dataset. By replacing the non-differentiable ϵ-insensitive term with Nesterov's smooth approximation f_μ, the dual becomes L-smooth and amenable to accelerated gradient methods. We derived a principled rule-of-thumb for μ that balances approximation error and gradient Lipschitzness, and justified a practical geometric decay schedule for μ that enables both rapid initial progress and asymptotic convergence to the true (unsmoothed) optimum. Under the chosen smoothing scheme the smoothed-dual objective attains the accelerated rate $O(1/k^2)$, while the unsmoothed primal–dual gap exhibits the expected slower decay in practice.

Empirical results confirm the theoretical rate for the smoothed dual across linear, polynomial and RBF kernels after a short kernel-dependent warm-up (the warm-up length grows with kernel complexity). The kernel Gram matrix spectral radius $\lambda_{\max}(K)$ materially affects the effective Lipschitz constant $L = \lambda_{\max}(K) + \epsilon/\mu$, explaining the different iteration counts observed for each kernel. The RBF kernel exhibits the longest warm-up and the largest sensitivity to μ-decay, consistent with its higher curvature and conditioning properties.

When focusing on optimization quality (objective values, duality gaps and feasibility), both approaches produce essentially equivalent dual objectives. The QP solver (OSQP via CVXPY) consistently reports extremely small solver-native residuals and update norms, reflecting its conservative enforcement of KKT conditions.

This project demonstrates that smoothed-gradient techniques, when combined with careful μ selection and projection onto the feasible set, yield a theoretically sound and practically competitive method for solving the SVR dual. The Nesterov-smoothed solver attains the expected accelerated convergence behavior and produces solutions that are essentially equivalent (in objective value) to a mature QP solver, while leaving open an attractive path for future work on scaling, conditioning, and standardized solver comparison metrics.

Chapter 6

Appendices

6.1 Proof that $f'_\mu(x)$ is Lipschitz continuous with constant $\frac{1}{\mu}$

To prove that $f'_\mu(x)$ is Lipschitz continuous with constant $\frac{1}{\mu}$, we consider the piecewise definition of the smoothed function $f_\mu(x)$, given in (1.23), and repeated here for clarity:

$$f_\mu(x) = \begin{cases} \frac{x^2}{2\mu} & \text{if } |x| \le \mu \\ |x| - \frac{\mu}{2} & \text{if } |x| > \mu \end{cases}$$

This function is continuous and piecewise differentiable. Its derivative is:

$$f'_\mu(x) = \begin{cases} \frac{x}{\mu} & \text{if } |x| < \mu \\ 1 & \text{if } x > \mu \\ -1 & \text{if } x < -\mu \end{cases}$$

We aim to show that for all $x_1, x_2 \in \mathbb{R}$:

$$|f'_\mu(x_1) - f'_\mu(x_2)| \le \frac{1}{\mu}|x_1 - x_2|$$

We distinguish three cases.

Case 1: $x_1, x_2 \in (-\mu, \mu)$

In this interval, $f'_\mu(x) = \frac{x}{\mu}$. Then:

$$|f'_\mu(x_1) - f'_\mu(x_2)| = \left|\frac{x_1 - x_2}{\mu}\right| = \frac{1}{\mu}|x_1 - x_2|$$

So the Lipschitz constant in this region is exactly $\frac{1}{\mu}$.

Case 2: $x_1, x_2 \in \mathbb{R} \setminus [-\mu, \mu]$

Here, $f'_\mu(x) = \pm 1$, i.e., the derivative is constant on each side. Then:

$$|f'_\mu(x_1) - f'_\mu(x_2)| = \begin{cases} 0 & \text{if } x_1, x_2 > \mu \text{ or } x_1, x_2 < -\mu \\ 2 & \text{if } x_1 > \mu \text{ and } x_2 < -\mu \text{ or vice versa} \end{cases}$$

If $f'_\mu(x_1) \neq f'_\mu(x_2)$, then $x_1 > \mu$, $x_2 < -\mu$, and:

$$|x_1 - x_2| \geq 2\mu \quad \Rightarrow \quad \frac{|f'_\mu(x_1) - f'_\mu(x_2)|}{|x_1 - x_2|} \leq \frac{2}{2\mu} = \frac{1}{\mu}$$

So the Lipschitz condition holds also in this case.

Case 3: One point inside $[-\mu, \mu]$, the other outside

Without loss of generality, suppose $x_1 \in [-\mu, \mu]$ and $x_2 > \mu$. Then:

$$f'_\mu(x_1) = \frac{x_1}{\mu} \in [-1, 1], \quad f'_\mu(x_2) = 1$$

So:

$$|f'_\mu(x_1) - f'_\mu(x_2)| = \left|\frac{x_1}{\mu} - 1\right| = \frac{|\mu - x_1|}{\mu}$$

Since $x_1 \leq \mu \leq x_2$, we have:

$$|x_1 - x_2| \geq |\mu - x_1| \Rightarrow |f'_\mu(x_1) - f'_\mu(x_2)| \leq \frac{1}{\mu}|x_1 - x_2|$$

The same argument holds for $x_2 < -\mu$.

In the end, in all cases, we have:

$$|f'_\mu(x_1) - f'_\mu(x_2)| \leq \frac{1}{\mu}|x_1 - x_2| \quad \forall x_1, x_2 \in \mathbb{R}$$

Hence, the function $f'_\mu(x)$ is Lipschitz continuous with constant $\frac{1}{\mu}$.

6.2 Lipschitz Continuity of the Linear Term $\beta \mapsto K\beta$

Let $K \in \mathbb{R}^{n\times n}$ be a symmetric positive semidefinite matrix, which is the case for all valid kernel matrices. Consider the linear transformation $T(\beta) = K\beta$. We aim to prove that T is Lipschitz continuous with constant $\|K\|$, where $\|K\|$ denotes the operator norm of K.

Operator Norm and Spectral Properties

The operator norm of a matrix K, induced by the Euclidean norm, is defined as:

$$\|K\| = \sup_{\|x\|=1} \|Kx\|$$

Since K is symmetric, the spectral theorem ensures that it is diagonalizable with real eigenvalues. Moreover, being positive semidefinite implies all eigenvalues are non-negative. Therefore, the operator norm coincides with the largest eigenvalue:

$$\|K\| = \lambda_{\max}(K)$$

This can also be seen from the Rayleigh quotient:

$$\lambda_{\max}(K) = \max_{\|x\|=1} x^\top K x$$

and noting that:

$$\|Kx\|^2 = x^\top K^\top K x = x^\top K^2 x$$

For symmetric K, K^2 shares the same eigenvectors, and the norm is maximized along the principal eigenvector.

Lipschitz Bound

Let $\beta_1, \beta_2 \in \mathbb{R}^n$, and define $v = \beta_1 - \beta_2$. Then:

$$\|K(\beta_1 - \beta_2)\| = \|Kv\| \leq \|K\| \cdot \|v\| = \|K\| \cdot \|\beta_1 - \beta_2\|$$

This confirms that the mapping $\beta \mapsto K\beta$ is Lipschitz continuous with constant $\|K\|$, which measures the maximal amplification factor that the matrix K can apply to any vector.

$$\|K(\beta_1 - \beta_2)\| \leq \|K\| \cdot \|\beta_1 - \beta_2\|$$

This result is instrumental in bounding the Lipschitz constant of the full gradient $\nabla Q_\mu(\beta)$, as shown in the main text.

6.3 Concave–Convex Reduction

In our SVR training, we wish to solve:

$$\max_{\beta \in Q} Q_\mu(\beta)$$

where $Q \subset \mathbb{R}^n$ is a closed convex set and Q_μ is smooth and concave. In order to apply Nesterov's accelerated scheme, which is formulated for **minimizing** a smooth, convex function, we observe the following facts:

6.3.1 Concavity vs. convexity

A twice-differentiable function $f : Q \to \mathbb{R}$ is **concave** if and only if its negative $-f$ is **convex**.
Concretely,

$$f(\theta x+(1-\theta)y) \;\geq\; \theta f(x)+(1-\theta)f(y) \quad\iff\quad -f(\theta x+(1-\theta)y) \;\leq\; -\big[\theta f(x)+(1-\theta)f(y)\big].$$

6.3.2 Smoothness preserved under negation

If f is L-smooth on Q, that is

$$\big\|\nabla f(x) - \nabla f(y)\big\| \;\le\; L\,\|x-y\| \quad \forall\, x,y \in Q,$$

then $-f$ is also L-smooth, since

$$\nabla\big(-f\big)(x) - \nabla\big(-f\big)(y) = -\big[\nabla f(x) - \nabla f(y)\big],$$

taking norms preserves the same Lipschitz bound.

6.3.3 Equivalence of argmax and argmin

Maximizing f over Q is equivalent to minimizing $-f$:

$$\arg\max_{x\in Q} f(x) \;=\; \arg\min_{x\in Q}\big[-f(x)\big].$$

Moreover, the optimal values satisfy

$$\max_{x\in Q} f(x) = -\min_{x\in Q}\big[-f(x)\big].$$

6.3.4 Algorithmic implications

Nesterov's accelerated gradient method guarantees, for a convex, L-smooth objective h on Q,

$$h(y^k) - h(y^*) \;=\; O\big(1/k^2\big),$$

where y^* minimizes h. By setting

$$h(\beta) = -\,Q_\mu(\beta),$$

all assumptions (convexity of h, smoothness of ∇h, feasibility domain Q) are satisfied. Therefore one may directly run the "minimization" form of Nesterov's algorithm on $-Q_\mu$ and recover the maximizer of Q_μ.

Bibliography

[1] Steven Diamond and Stephen Boyd. "CVXPY: A Python-embedded modeling language for convex optimization". In: *Journal of Machine Learning Research* 17.83 (2016), pp. 1–5.

[2] Yurii Nesterov. "Smooth minimization of non-smooth functions". In: *Mathematical Programming* (2005), pp. 127–152. DOI: `10.1007/s10107-004-0552-5`.

[3] Weiran Wang and Miguel A Carreira-Perpinán. "Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application". In: *arXiv preprint arXiv:1309.1541* (2013).

[4] Akshay Agrawal et al. "A rewriting system for convex optimization problems". In: *Journal of Control and Decision* 5.1 (2018), pp. 42–60.

[5] Daniel Gabay and Bertrand Mercier. "A dual algorithm for the solution of nonlinear variational problems via finite element approximation". In: *Computers & Mathematics with Applications* 2.1 (1976), pp. 17–40.

[6] Roland Glowinski and Alain Marroco. "Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de Dirichlet non linéaires". In: *ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique* 9.R2 (1975), pp. 41–76.

[7] Peter J Rousseeuw and Annick M Leroy. *Robust regression and outlier detection.* John wiley & sons, 2003.

List of Figures