Contents

Project 10 - Description

(M) is a SVR-type approach of your choice (in particular, with one or more kernels of your choice).

(A1) is an algorithm of the class of level bundle methods, 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.

Use of an off-the-shelf solver for the Master Problem of the bundle method is allowed.

Introduction

SVR and Level Bundle Method

This project focuses on the development and implementation of an SVR (Support Vector for Regression) capable of learning from a dataset in the form of "feature x target," where "target" must be a vector of dimensions , in accordance with the definition of SVR. In addition to the basic implementation, a significant part of the SVR will leverage the Level Bundle Method for optimizing the dual function.

For the Master Problem of the **SVR with LBM**, the MATLAB function `quadprog` was used. This function is primarily designed for solving quadratic objective functions with linear terms.

Instead, for the **general purpose SVR** the `fmincon` function was used since this function can solve non-linear problems, making it ideal for our use case.

Support Vector for Regression (SVR)

Support Vector Regression (SVR) aims to find a function $f(X)$ that approximates the training data while minimizing a given loss function. The primal optimization problem is formulated as follows:

$$\begin{aligned} \min_{w,b,\xi,t^*} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ \text{s.t.:} \quad & \\ & y_i - \langle w, x_i\rangle - b \leq \varepsilon + \xi_i, \\ & \langle w, x_i\rangle + b - y_i \leq \varepsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \geq 0 \end{aligned}$$

where:

- w and b define the regression hyperplane
- ξ_i, ξ_i^* are slack variables that account for deviations beyond the margin ε
- C is a regularization parameter

By applying Lagrange multipliers and transforming the problem into its dual

formulation, we obtain:

$$\max_{\alpha} \quad \sum_{i=1}^{n} y_i \alpha_i - \varepsilon \sum_{i=1}^{n} |\alpha_i| - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

subject to:

$$\sum_{i=1}^{n} \alpha_i = 0,$$
$$-C \leq \alpha_i \leq C \quad \forall i = 1, \ldots, n.$$

where $K(x_i, x_j)$ is a kernel function that allows the method to handle non-linear relationships by implicitly mapping the input data to a higher-dimensional space.

Once the dual problem is solved, the **support vectors** are identified as the data points corresponding to nonzero Lagrange multipliers. Specifically, the support vectors are found by selecting the indices (i, j) that satisfy the condition:

$$\{(i, j) \mid |\alpha_i| > \tau\}$$

where τ is a small positive threshold to account for numerical precision. These support vectors are the most influential data points in defining the regression function, as they determine the final predictive model.

Overview of the Level Bundle Method

The Level Bundle Method (LBM) is an optimization approach that refines solutions iteratively by leveraging cutting-plane techniques and a level constraint. It is particularly useful in non-differentiable optimization problems, such as those encountered in support vector regression.

Reformulating the Problem for `quadprog` in MATLAB

Since the SVR function defined before is **non-differentiable**, we apply the **Level Bundle Method** to approximate it iteratively. Especially we find a new solution by solving the LBM objective function defined as follow:

$$\alpha_{k+1} = \arg\min_{\alpha} \left\{ \frac{1}{2} \|\alpha - \hat{\alpha}_k\|^2 \,\middle|\, \hat{f}_k(\alpha) \leq f_k^{\text{level}},\ \alpha \in X \right\}$$

where:

- α represents the vector of dual variables in the **Support Vector Regression (SVR)** problem.

- $\hat{\alpha}_k$ is the best solution found so far at iteration k.

- f_k^{level} is the current level used to restrict the search space within an acceptable region.
- X is the set of original constraints of the SVR dual problem.
- $\hat{f}_k$ is defined as follow:

$$\hat{f}_k(x) := \max_{j \in \mathcal{B}_k} \{f(x_j) + \langle \xi_j, x - x_j \rangle\}$$

Where:

$$f(x) = \frac{1}{2} x^\top K x + \varepsilon \sum_{i=1}^{n} |x_i| - y^\top x$$

$$\xi = Kx + \epsilon \cdot \text{sign}(x) - y$$

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x > 0, \\ -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0. \end{cases}$$

However, from MATLAB documentaion, the `quadprog` general function is designed to solve quadratic optimization problems in the following standard form:

$$\begin{aligned} \min_x \quad & \frac{1}{2} x^\top H x + f^\top x \\ \text{s.t.} \quad & Ax \leq b, \\ & A_{\text{eq}} x = b_{\text{eq}}, \\ & lb \leq x \leq ub. \end{aligned}$$

Therefore, it is necessary to reformulate the **Level Bundle Method** problem so that it is compatible with the form required by `quadprog`.

Quadratic Norm Expansion

The objective function of the Level Bundle Method can be rewritten explicitly as follows:

$$\frac{1}{2}\|\alpha - \hat{\alpha}_k\|^2 = \frac{1}{2}\left(\alpha^\top \alpha - 2\hat{\alpha}_k^\top \alpha + \hat{\alpha}_k^\top \hat{\alpha}_k\right) = \frac{1}{2}\alpha^\top \alpha - \hat{\alpha}_k^\top \alpha + \frac{1}{2}\hat{\alpha}_k^\top \hat{\alpha}_k.$$

Since the constant term $\frac{1}{2}\hat{\alpha}_k^\top \hat{\alpha}_k$ does not affect the minimization, we can omit it. The function to minimize then becomes:

$$\frac{1}{2}\alpha^\top \alpha - \hat{\alpha}_k^\top \alpha.$$

Reformulation into `quadprog` Format

By comparing this expression with the standard objective function solved by `quadprog`:

$$\frac{1}{2}x^\top Hx + f^\top x,$$

we obtain the following parameters:

- $H = I$ (identity matrix, since the quadratic term is $\frac{1}{2}x^\top x$).
- $f = -\hat{x}_k$ (since the linear term is $-\hat{\alpha}_k^\top \alpha$).

Substituting these values, we obtain:

$$\frac{1}{2}x^\top Hx + f^\top x = \frac{1}{2}x^\top x - \hat{x}_k^\top x.$$

That is exactly the function that we want to minimize.

Constraints

The Level Bundle Method algorithm is subject to the following constraints:

$$\begin{aligned}
&\text{s.t. :} \\
&\quad \hat{f}_k(\alpha) \leq f_k^{\text{level}} \\
&\quad \sum_{i=1}^{n} \alpha_i = 0 \\
&\quad -C \leq \alpha_i \leq C \quad \forall i = 1, \ldots, n
\end{aligned}$$

Thus, they must be rewritten into a form solvable by `quadprog`, just as we did for the objective function.

Bundle Cuts Constraint

Given the following constraint:

$$\hat{f}_k(\alpha) \leq f_k^{\text{level}}$$

We can implement it through the cutting plane approach, where the function is approximated via a collection of tangent hyperplanes. These hyperplanes are defined by the subgradients of the function and translate into linear constraints of the form:

$$f(\alpha_k) + \langle \xi_k, \alpha - \alpha_k \rangle \leq f^{\text{level}}$$

Where ξ_j are the subgradients computed at those points and $f(\hat{\alpha_k})$ is the real objective value evaluated.

This formulation must be rewritten in the form $Ax \leq b$. Rewriting the constraint:

$$f(\alpha_k) + \langle \xi_k, \alpha - \alpha_k \rangle \leq f^{\text{level}} \iff \langle \xi_k, \alpha \rangle - f^{\text{level}} \leq \langle \xi_j, \alpha_k \rangle - f(\alpha_k)$$

With:

$$A = \begin{bmatrix} \xi_k^\top & -1 \end{bmatrix}, \quad b = \langle \xi_k, \hat{\alpha_k} \rangle - f(\hat{\alpha_k})$$

Note that the current level f_k^{level} is updated at each iteration using the following formula:

$$f_{\text{level}} = \theta f + (1 - \theta) f_{\text{best}}$$

Equality Constraint

$$\sum_{i=1}^{n} \alpha_i = 0.$$

This can be defined directly as:

$$A_{\text{eq}} = \begin{bmatrix} \mathbf{1}^\top \end{bmatrix}, \quad b_{\text{eq}} = 0.$$

Bound Constraints

$$-C \leq \alpha_i \leq C,$$

These can be defined as:

$$lb = [-C, -\infty],$$
$$ub = [C, f_{\text{level}}].$$

General Notes and Considerations

As previously described, this project aims to implement an SVR that leverages the Level Bundle Method for optimization. However, the goal is not to achieve a fully optimized SVR in terms of hyperparameters or generalization performance on the dataset cause this would require additional tuning techniques such as grid search or k-fold cross-validation, which are beyond the scope of this project.

Nevertheless, we will present the error achieved using the Mean Squared Error (MSE) metric, along with the selected hyperparameters. Additionally, we will compare our implementation against SVR general solver (A1), which will be used as an oracle.

Conclusion

We have demonstrated how the objective function of the **Level Bundle Method** can be rewritten in the standard form required by `quadprog`. In the following sections, we will present the implementation of the algorithm following the steps outlined so far.

Implementation

Dataset

Initially, we used simple functions such as sine, exponential, and step functions with added noise. These synthetic datasets allowed us to test and verify the correctness of our SVR implementation in terms of predictions and performance. Subsequently, we moved to a larger and more complex dataset: the **Abalone dataset**. This is a regression dataset where the goal is to predict the age of abalones based on various features.

The dataset is structured as follows:

- The first 8 columns correspond to the input features.
- The last column represents the target value (age of the abalone).

Before training, the features are normalized using `zscore`:

```
[X, y] = training_data("abalone"); % Alternatively: sin, exp ..
X = zscore(X);
```

Next, we define the SVR dual function, which computes both the function value `f` and the corresponding subgradient `g`:

```
function [f, g] = svr_dual_function(x, K, y, epsilon)
    f = 0.5 * x' * (K * x) + epsilon * sum(abs(x)) - y' * x;
    g = K * x + epsilon * sign(x) - y;
end
```

(A2) SVR general-purpose solver

For the implementation of a generic SVR solver, we formulated the dual problem using MATLAB's lambda functions. Unlike quadprog, fmincon does not require explicit Hessian matrix definitions—a key advantage that significantly improves scalability for large-scale problems

```
% defining the lambda function
svr_dual =  @(x) svr_dual_function(x, K, Y, epsilon);

% Starting point
alpha0 = zeros(n, 1);

% Inequality constraints
A = [];
b = 0;

% Equality constraint: sum(alpha) = 0
Aeq = ones(1, n);
beq = 0;

% Lower and upper bounds: 0 <= alpha <= C
lb = -C * ones(n, 1);
ub = C * ones(n, 1);

% Solve the non-linear quadratic problem using fmincon
options = optimoptions('fmincon', 'Display', 'iter', ...
    'SpecifyObjectiveGradient', true, 'MaxIterations', maxIter);

alpha = fmincon(svr_dual, alpha0, A, b, Aeq, beq, lb, ub, [], options);
```

Once `fmincon` has completed execution, we extract the **support vectors** from the solution.

```
% Identify support vectors based on tolerance threshold
sv_indices = find(abs(alpha) > tol);

% Compute bias for the prediction phase
bias = mean(Y(sv_indices) - K(sv_indices, :) * alpha);
```

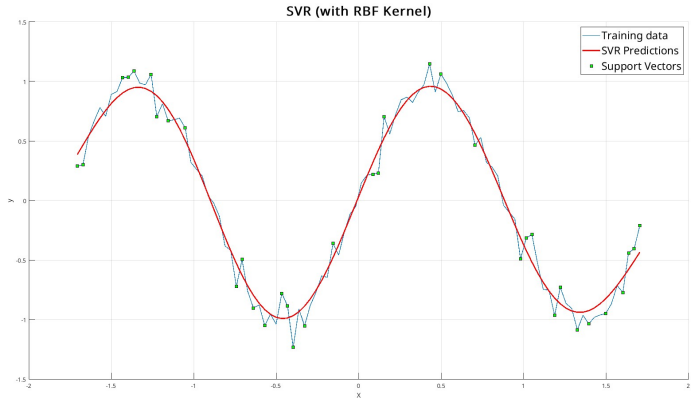
Performance Evaluation

We evaluated this SVR implementation on both synthetic datasets and the **Abalone dataset**. While the model generalizes well on synthetic data, its performance deteriorates significantly on the Abalone dataset due to the **high number of constraints**.

Synthetic data

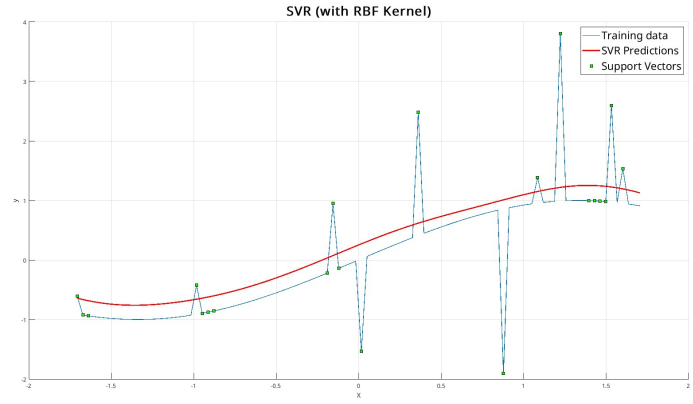
- **Sine function**

$$Y = \sin(3X) + 0.1 \cdot \mathcal{N}(0,1).$$



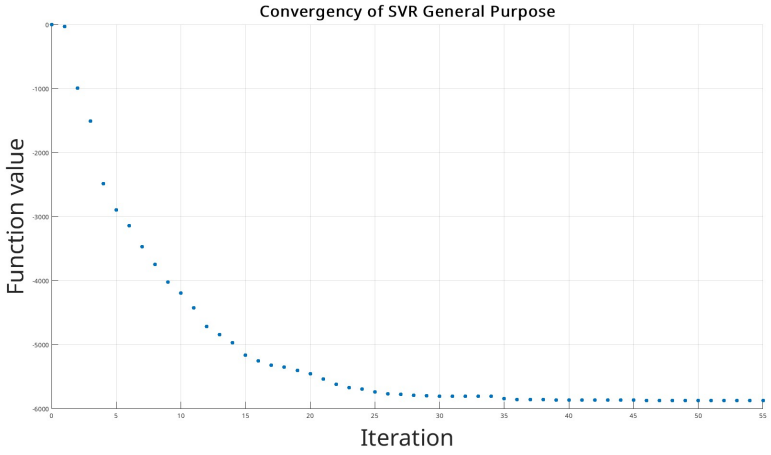
- **Outliers test**

$$Y = \sin(X) + \underbrace{2 \cdot \mathcal{N}(0,1)}_{\text{10 random points}} .$$



Abalone

- **MSE**: 4.2187



(A1) Level Bundle Method Implementation

Previously, we mathematically formulated the objective function and constraints of the **Level Bundle Method (LBM)** in a format compatible with `quadprog`. We now translate these formulations into MATLAB code, ensuring a direct correspondence between the mathematical expressions and their implementation. Firstly we define the solver signature function as:

```
function alpha_opt = mp_solve(alpha_hat, bundle, f_level, C)
```

Where:

- `alpha_hat` is the current α
- `bundle` is a structure containing:
  - **bundle.alpha** $\rightarrow$ vector of dual variables α
  - **bundle.g** $\rightarrow$ vector of the subgradients
  - **bundle.f** $\rightarrow$ vector of function evaluations
- `f_level` is the current level

Hessian Matrix and Linear Coefficients

The objective function of the optimization problem is defined as:

$$H = I, \quad f = -\hat{x}_k.$$

In MATLAB, this is implemented as:

```
H = blkdiag(eye(n), 0); % n = length(alpha_hat)
f = [-alpha_hat; 0];
```

Inequality Constraints

The linear constraints are represented in matrix form as:

$$A = \begin{bmatrix} \xi_k^\top & -1 \end{bmatrix}, \quad b = \langle \xi_k, \hat{\alpha_k} \rangle - f(\hat{\alpha_k})$$

Translated into MATLAB:

```
A = [bundle.g' -ones(m, 1)]; % m = length(bundle.f)
b = sum(bundle.g .* bundle.alpha, 1)' - bundle.f';
```

Equality Constraints

The equality constraints are given by:

$$A_{\text{eq}} = \left[\mathbf{1}^\top\right], \quad b_{\text{eq}} = 0.$$

MATLAB implementation:

```
Aeq = [ones(1, n) 0];
beq = 0;
```

Variable Bounds

The variables are subject to the following bounds:

$$lb = [-C, -\infty],$$
$$ub = [C, f_{\text{level}}].$$

Implemented as:

```
lb = [-C * ones(n, 1); -inf];
ub = [C * ones(n, 1); f_level];
```

Solving the Optimization Problem

Finally, we use `quadprog` to find the optimal solution. So the ending function is:

```
function alpha_opt = mp_solve(alpha_hat, bundle, f_level, C)
    n = length(alpha_hat);
    m = length(bundle.f);

    H = blkdiag(eye(n), 0);
    f = [-alpha_hat; 0];

    A = [bundle.g' -ones(m, 1)];
    b = sum(bundle.g .* bundle.alpha, 1)' - bundle.f';

    Aeq = [ones(1, n) 0];
    beq = 0;

    lb = [-C * ones(n, 1); -inf];
    ub = [C * ones(n, 1); f_level];

    options = optimoptions('quadprog','Display','off');

    [sol, ~, exitflag] = quadprog(H, f, A, b, Aeq, beq, lb, ub, [], options);

    if exitflag <= 0
        warning("best solution not found, keeping prev...");
        alpha_opt = alpha_hat;
    else
        alpha_opt = sol(1:n);
    end
end
```

LBM Algorithm: Pseudo-code Implementation

```
Input:
    K           % kernel matrix
    alpha       % current solution (dual vector)
    C           % upper bound on dual variables
    theta       % level parameter
    tol         % convergence tolerance
    max_iter    % maximum number of iterations

% Initialization
[f, g] = svr_dual_function(alpha)
f_best = f

bundle.alpha = alpha
bundle.f = f
bundle.g = g

for iter = 1 to max_iter:
    % Compute the acceptance level
    level = theta * f + (1 - theta) * f_best

    % Solve the master problem to get the new point
    alpha_new = mp_solve(alpha, bundle, level, C)

    % Evaluate the function and subgradient at alpha_new
    [f_new, g_new] = svr_dual_function(alpha_new, K, y, epsilon)

    % Check if the new point is acceptable
    if f_new < f_best:
        f_best = f_new

    if f_new <= level:
        alpha = alpha_new
        f = f_new

    % Update bundle
    bundle.alpha = [bundle.alpha, alpha_new]
    bundle.f = [bundle.f, f_new]
    bundle.g = [bundle.g, g_new]

    % Check for convergence
    if ||alpha_new - alpha|| < tol:
        break

return alpha  % Return the optimal solution
```

Performance Evaluation

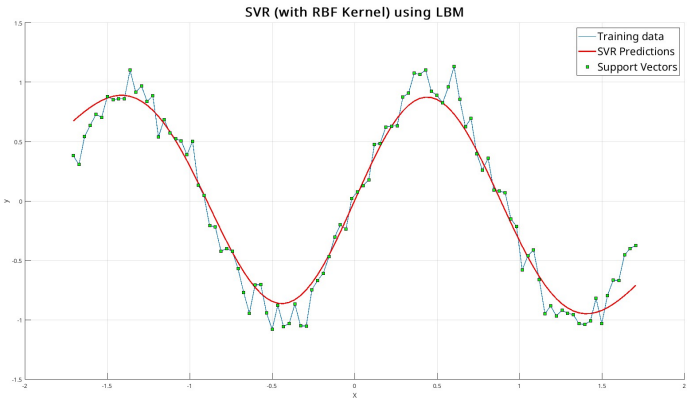
SVR Training with LBM on Synthetic Data

To evaluate the performance of our SVR (Support Vector Regression) model implemented with the Level Bundle Method (LBM), we conducted preliminary testing on predefined synthetic datasets. The results demonstrate that the model retains the generalization capability characteristic of classical SVR while maintaining comparable computational efficiency in terms of execution time.

Below, we present the predictive performance on the sine function as a representative example of the model's generalization capabilities. The remaining functions (omitted for brevity) demonstrate behaviors fully aligned with the standard SVR implementation.

- **Sine function**

$$Y = \sin(3X) + 0.1 \cdot \mathcal{N}(0, 1).$$



Training SVR with LBM on the Abalone Dataset

As mentioned earlier, we used the Abalone dataset to evaluate the actual performance of our SVR implementation. This dataset contains 4177 samples with 8 features each, making it a perfect candidate as it introduces challenges such as:

- A high number of constraints

- Increased memory consumption
- Prolonged training time due to repeated calls to the solver

High Number of Constraints The main issue encountered in the implementation of the **Level bundle method** in MATLAB is the uncontrolled growth of the bundle size at each iteration.

Every cycle of the algorithm adds new constraints to the system, leading to an **exponential increase in the number of conditions** to be handled. This progressive expansion of the bundle has two critical effects:

1. **Memory overload**: The bundle data structure, especially with high-dimensional datasets, consumes resources in a drastically non-linear manner, making allocation unsustainable for large data volumes.
2. **Performance degradation of quadprog**: MATLAB's quadratic programming solver becomes progressively slower—sometimes even leading to total stalls—due to the need to process constraint matrices with thousands of rows.

To address this issue, we introduced the **bundle truncation** technique: once a certain threshold is exceeded, the oldest constraints are removed from the bundle, leaving only the most recent k constraints. This significantly improved both memory efficiency and computational performance.

```
if size(bundle.alpha, 2) > max_constraints
    bundle.alpha = bundle.alpha(:, 2:end);
    bundle.f = bundle.f(2:end);
    bundle.g = bundle.g(:, 2:end);
end
```

Memory Consumption Despite the introduction of bundle truncation, memory consumption remained significantly high. This was mainly due to the allocation of large matrices. For instance, given that the dataset contains 4177 features, the matrix H passed to `quadprog` has a size of 4178×4178, which is excessive for an identity matrix.

To mitigate this issue, we decided to switch to **sparse matrices**, which drastically reduced memory usage, as only nonzero elements are stored. Additionally, this optimization ensured that the `quadprog` solver could terminate within a reasonable time frame. So the new `mp_solve` function can be written as:

```
H = blkdiag(speye(n), 0);
f = sparse([-alpha_hat; 0]);
```

```
A = sparse([bundle.g' -ones(m, 1)]);
b = sparse(sum(bundle.g .* bundle.alpha, 1)' - bundle.f');

Aeq = sparse([ones(1, n) 0]);
beq = 0;

lb = sparse([-C * ones(n, 1); -inf]);
ub = sparse([C * ones(n, 1); f_level]);
```

Choosing the Solver: Is Quadprog Really the Best Option? The current implementation relies on **quadprog**, with a training time of approximately **300 seconds**, but alternative solvers could significantly improve time efficiency.

Further gains can be achieved by adopting **high-performance solvers** optimized for large-scale problems, as such alternatives leverage best algorithms and parallelization. This could reduce training times by **an order of magnitude** without compromising accuracy.

Conclusions

Hyperparameters Used

During the hyperparameter selection phase for the various SVR models, we maintained the same values as much as possible to ensure a fair and consistent comparison among the different approaches.

SVR Parameters:

- `kernel_function`: `RBFKernel()`

  Kernel function used in the Support Vector Regression (SVR) model (Radial Basis Function Kernel in this case).

- `C`: 1

  Regularization parameter for the SVR model, controlling the trade-off between model complexity and training error.

- `epsilon`: 0.05

  Epsilon margin for the SVR model, defining the tolerance for prediction errors.

- `max_iter`: 500

  Maximum number of iterations for the Level Bundle Method (LBM).

- `opt`: `lbm`

  Optimizer used for the SVR model (Level Bundle Method in this case).

LBM Parameters:

- `tol`: 1e-3

  Tolerance for the subgradient step in the LBM (stopping criteria).

- `theta`: 0.1

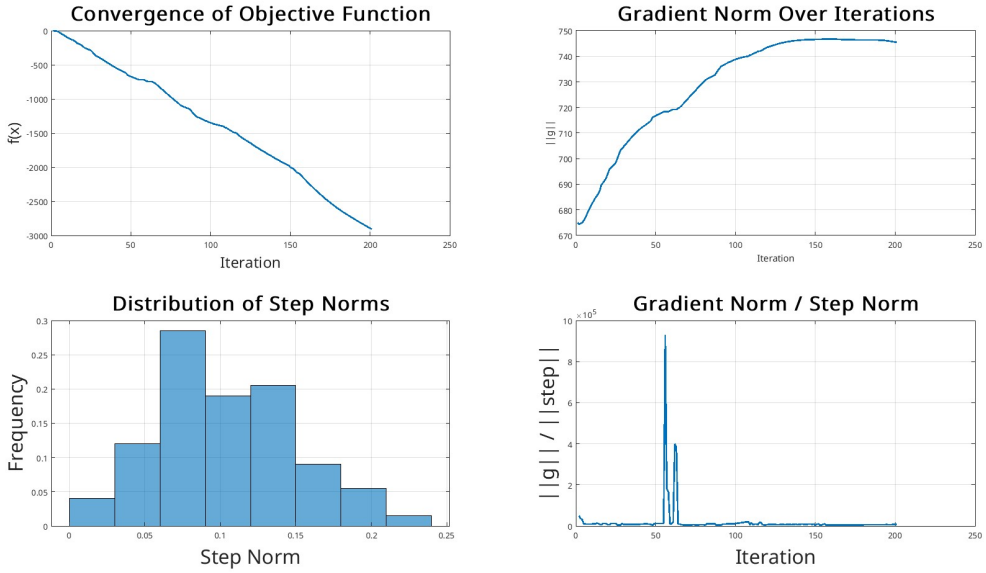
  Convex combination parameter used in the LBM.

- `max_constraints`: 50

  Maximum number of constraints allowed in the bundle.

Results

Thanks to the optimizations introduced so far, our LBM method achieved **sustainable performance**, with a **Mean Squared Error (MSE) of 4.3729**, and a **linear convergence rate**. This result is totally comparable with the Oracle that obtained an **MSE of 4.2187** as shown in the previous chapter.



From the analysis of the convergence plots of the SVR with the **Level Bundle method**, several significant conclusions can be drawn:

- The **objective function** plot shows a **steady and monotonic decrease** until approximately *-3000*, indicating that the optimization is effectively progressing toward minimization.
- The **gradient norm** stabilizes around *745* after about *150* iterations, suggesting the attainment of a **stationary point**.
- The **distribution of step norms** reveals that the algorithm prefers step sizes between *0.05* and *0.15*, with a higher concentration around *0.075*, indicating a balanced behavior between exploration and exploitation of the solution space.
- Particularly interesting is the relationship between **gradient norm** and **step norm**, which shows two significant peaks around iteration *50*, suggesting that during that phase, the algorithm traversed regions with **strong curvature** or **gradient discontinuities**.

Overall, the **Level Bundle method** demonstrates good computational efficiency, achieving convergence in approximately *150* iterations with a stable trend in the objective function, characteristics that make it well-suited for complex optimization problems such as those addressed in the SVR context.