Ecole Nationale Supérieure
des Techniques Avancées

Project Report

Cooperative Optimization for Data Science

Numerical Project in Python

Cooperative Kernel Regression

Ahmed Yassine Chraa, Emma de Charry, Pauline Zhou

Contents

List of Figures

1 Part I (Classes 1 and 2)

1.1 Parameters

Here are the parameters of the 4 requested algorithms in Part I, applied to the following problem:

$$\alpha^* = \min_{\alpha \in \mathbf{R}^m} \sum_{a=1}^{5} \frac{\sigma^2}{10} \alpha^T K_{mm} \alpha + \frac{1}{2} \sum_{i \in A} \|y_i - K_{(i)m}\alpha\|_2^2 + \frac{\nu}{10}\|\alpha\|^2 = \min_{\alpha \in \mathbf{R}^m} \sum_{a=1}^{N} f_a(\alpha)$$

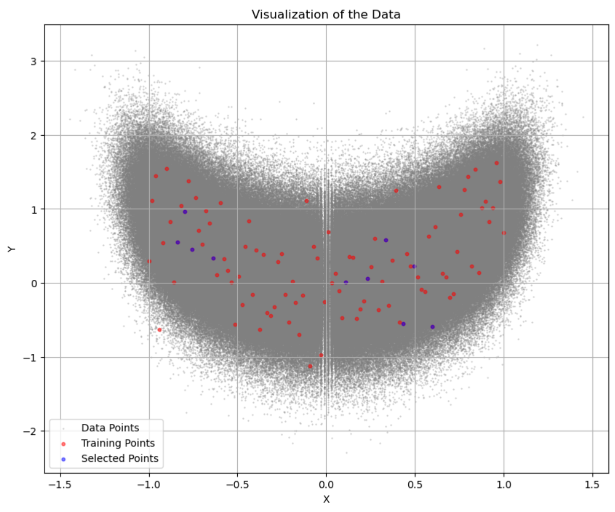
$$\begin{aligned} n &= 100 \\ m &= \sqrt{n} = 10 \\ a &= 5 \\ \nu &= 1.0 \\ \sigma &= 0.5 \\ W &= \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & 0 & 0 & \frac{1}{3} & \frac{1}{3} \end{bmatrix} \end{aligned}$$

f_a functions being convex, we have the following:

$$\nabla f_a(\alpha) = \frac{\sigma^2}{5} K_{mm}\alpha + (\sum_{i \in A} K_{(i)m}^T K_{(i)m})\alpha - \sum_{i \in A} K_{(i)m}^T y_i + \frac{\nu}{5}\alpha$$

$$\nabla^2 f_a(\alpha) = \frac{\sigma^2}{5} K_{mm} + \sum_{i \in A} K_{(i)m}^T K_{(i)m} + \frac{\nu}{5}\mathbf{I}$$

We first visualize the data:



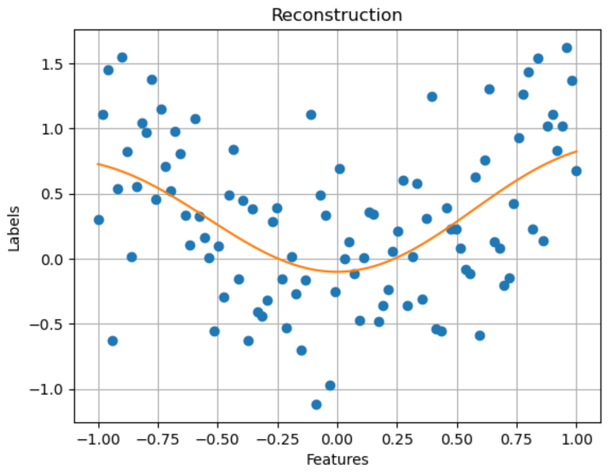


Figure 1: Visualization of the data

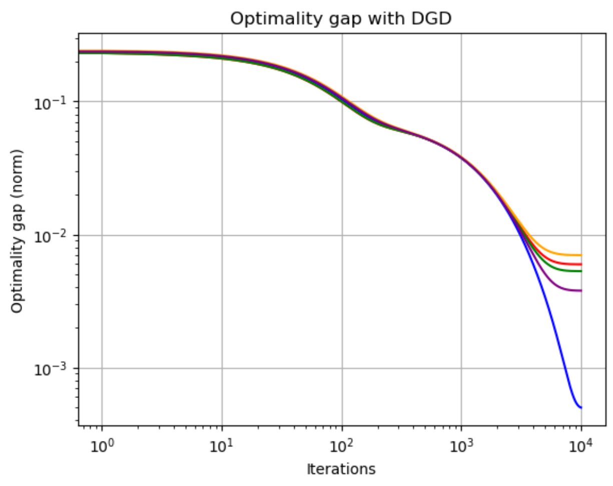
1.2 Decentralized gradient descent

Decentralized Gradient Descent (DGD) algorithm :

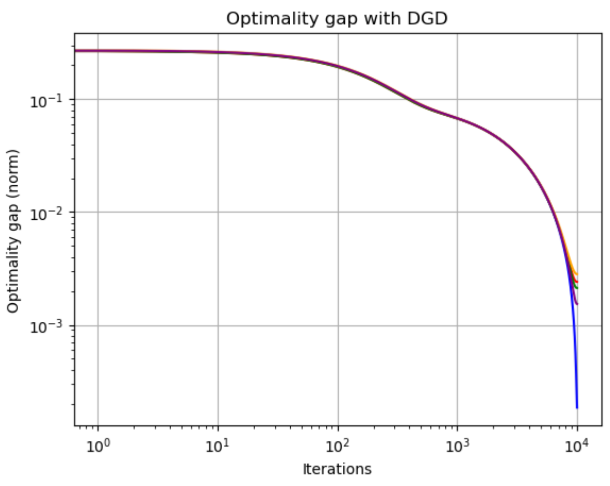
- Start with $\boldsymbol{\alpha}_0^i \in \mathbf{R}^n$ at each device i and a step size μ
- For all $k = 0, 1, \dots$:
  - Communication step: send and receive $\boldsymbol{\alpha}_k^i$ from your neighbors
  - Computation step: Each device i computes:

$$\boldsymbol{\alpha}_{k+1}^i = \sum_{j=1}^{N} w_{ij}\boldsymbol{\alpha}_k^j - \mu \nabla f_i\left(\boldsymbol{\alpha}_k^i\right)$$

We coded the DGD algorithm and ran it with `step_size = 0.002` and `step_size = 0.0009`. Here are the results we got:



(a) DGD with `step_size` $= 0.002$



(b) DGD with `step_size` $= 0.0009$

Figure 2: Convergence of DGD

Indeed, Theorem 2.1 tells us that DGD converges in $O(\frac{\mu}{1-\gamma})$ if μ is small enough. We have thus chosen μ according to the theory: $\mu = 0.0009$

1.3 Gradient tracking

Gradient Tracking algorithm :

- Initialize $\boldsymbol{\alpha}_0^i \in \mathbf{R}^n, \boldsymbol{g}_0^i \in \mathbf{R}^n$ at each device and a stepsize α
- For all $k = 0, 1, \ldots$
  - Communication step: send and receive $\boldsymbol{x}_k^i$ and $\boldsymbol{g}_k^i$ from your neighbors
  - Computation step: each device i computes:

$$\boldsymbol{\alpha}_{k+1}^i = \sum_{j=1}^{N} w_{ij} \boldsymbol{\alpha}_k^j - \mu \boldsymbol{g}_k^i$$

$$\boldsymbol{g}_{k+1}^i = \sum_{j=1}^{N} w_{ij} \boldsymbol{g}_k^j + \left(\nabla f_i\left(\boldsymbol{\alpha}_{k+1}^i\right) - \nabla f_i\left(\boldsymbol{\alpha}_k^i\right)\right)$$

Likewise, here are the results for the gradient tracking algorithm:

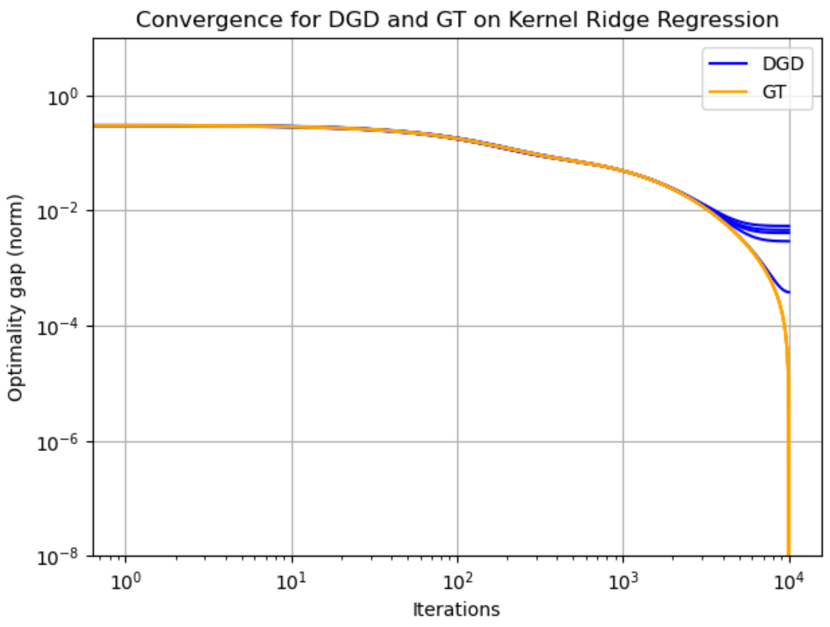


Figure 3: GT vs DGD, `step_size` $= 0.002$

The optimality gap curves indicate that both methods exhibit similar behavior in the initial stages, with comparable convergence rates. However, as iterations progress, DGD demonstrates noticeable variance among agents, leading to a slower convergence rate. In contrast, GT maintains a more consistent trajectory, ultimately reaching a lower optimality gap more efficiently. This highlights the advantage of GT in mitigating information drift across the network, making it preferable for achieving consensus and faster convergence in cooperative optimization settings.

1.4 Dual decomposition and ADMM

Peer-to-peer dual decomposition :

- Initialize $\lambda_0^{ij} \in \mathbf{R}^n$ for $(i,j) \in \mathcal{E}, j < i$, and interpret λ^{ij} for $j > i$ as λ^{ji}.
- For $k = 0, 1, \ldots$ iterate:
  - Computation step: For all devices i do,

$$\boldsymbol{\alpha}^{i,*}(\lambda_k) := \arg\min_{\boldsymbol{\alpha}_i} \left\{ f_i\left(\boldsymbol{\alpha}^i\right) + \sum_{i\sim j \equiv j \in N_i} (-1)^a \lambda_k^{ij,\top} \boldsymbol{\alpha}^i \right\}$$

where

$$a = \begin{cases} 0 & \text{if } i > j \\ 1 & \text{otherwise} \end{cases}$$

  - Communication step: Send $\boldsymbol{\alpha}^{i,*}(\lambda_k)$ to neighbors j
  - Computation step: Update for node i, j :

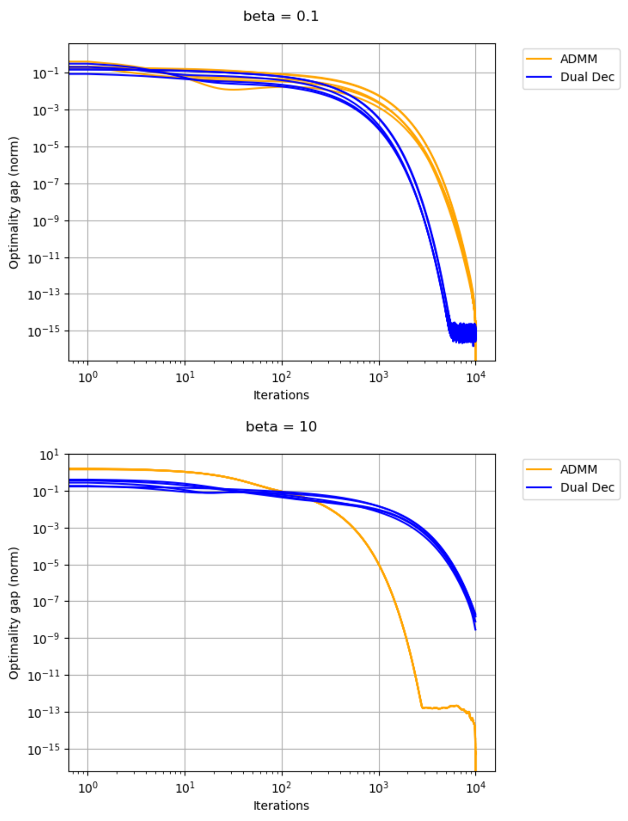
$$\lambda_{k+1}^{ij} = \lambda_k^{ij} + \mu\left[\boldsymbol{\alpha}^{i,*}(\lambda_k) - \boldsymbol{\alpha}^{j,*}(\lambda_k)\right], \quad j < i$$

ADMM (general case) :

- Start with an initial value for $\boldsymbol{x}_0, \boldsymbol{y}_0, \lambda_0$ and a positive scalar β,
- Iterate:

$$\boldsymbol{\alpha}_{k+1} = \arg\min_{\boldsymbol{\alpha}\in\mathbf{R}^n} \mathcal{L}_\beta(\boldsymbol{\alpha}, \boldsymbol{y}_k, \lambda_k)$$
$$\boldsymbol{y}_{k+1} = \arg\min_{\boldsymbol{y}\in\mathbf{R}^p} \mathcal{L}_\beta(\boldsymbol{\alpha}_{k+1}, \boldsymbol{y}, \lambda_k)$$
$$\lambda_{k+1} = \lambda_{k+1} + \beta\left(A\boldsymbol{\alpha}_{k+1} + B\boldsymbol{y}_{k+1} - c\right)$$

Theorems 4.1 & 4.2: Dual decomposition converges linearly if $\mu < 2m/\sigma_{max}^2(A)$. The Dual decomposition plot presented is drawn with $\mu = 0.1$



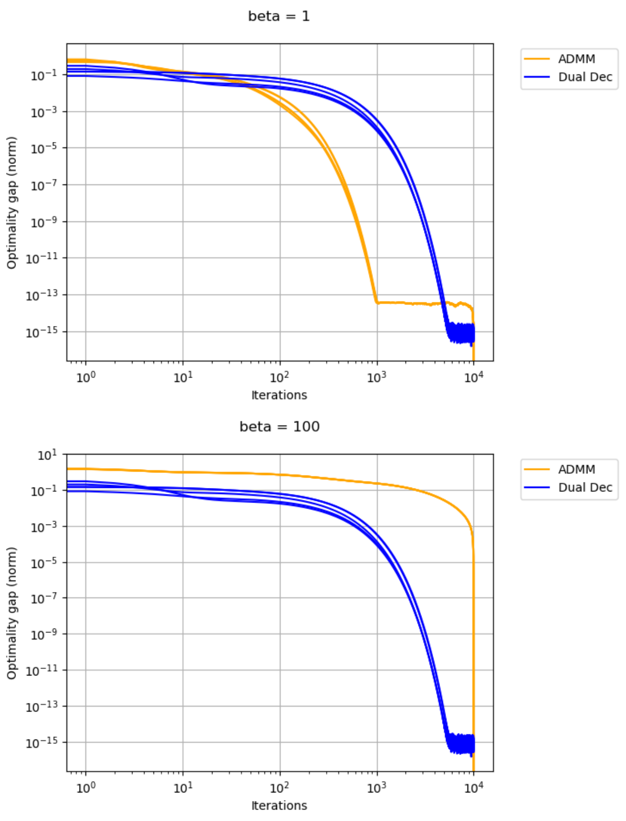
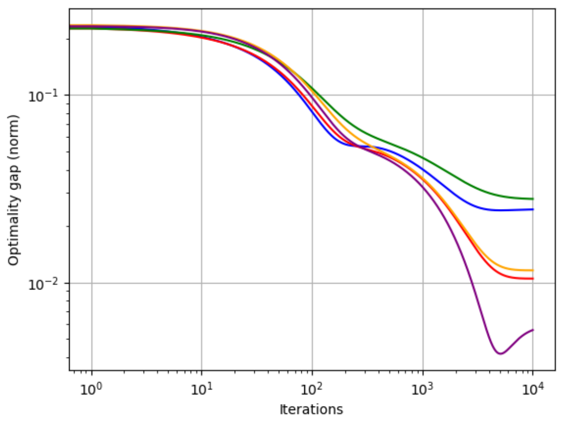


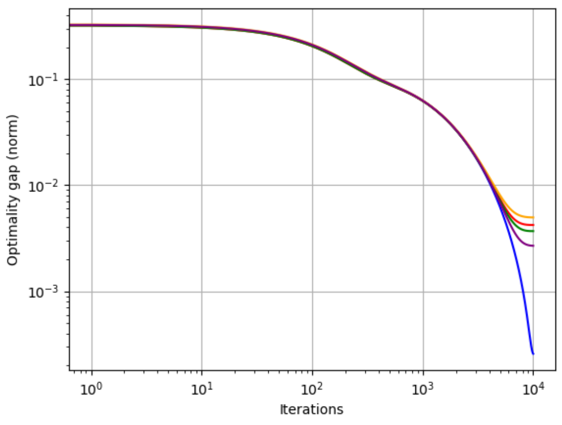
Figure 4: Convergence of Dual decomposition vs ADMM

The performance of ADMM and Dual Decomposition was analyzed for different values of the regularization parameter β. The optimality gap, plotted on a log-log scale, shows that both methods achieve high precision, but their convergence behavior varies. For small values of β (0.1), Dual Decomposition exhibits slightly faster convergence, maintaining stability across iterations. However, as β increases $(1, 10)$, ADMM outperforms Dual Decomposition in convergence speed, achieving machine precision ($\approx 10^{-15}$) in the case of $\beta = 10$. Finally, for $\beta = 100$ Dual Decomposition converges faster, but ADMM seems to attain a better approximation. This suggests that ADMM benefits more from stronger coupling across agents, whereas Dual Decomposition remains competitive in lower regularization regimes. Overall, the choice of method depends on the problem structure and the desired trade-off between convergence speed and stability. As written in the lecture notes, ADMM is superior when β is chosen carefully.

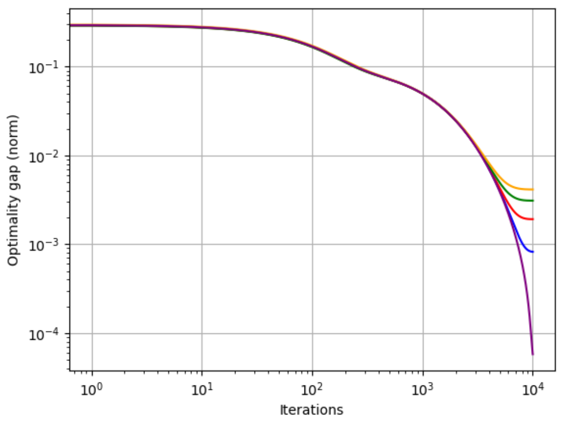
1.5 Different graphs



(a) DGD with Line Graph

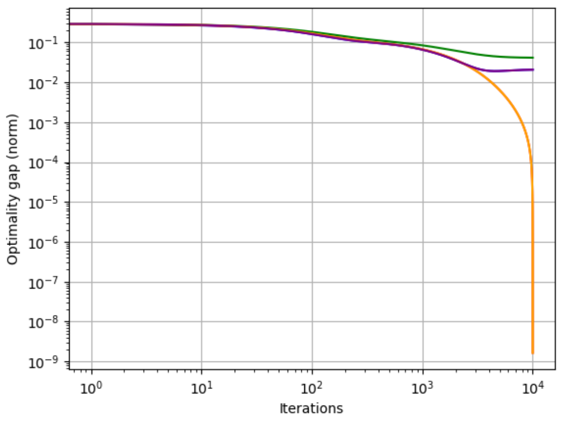


(b) DGD with Small World Graph

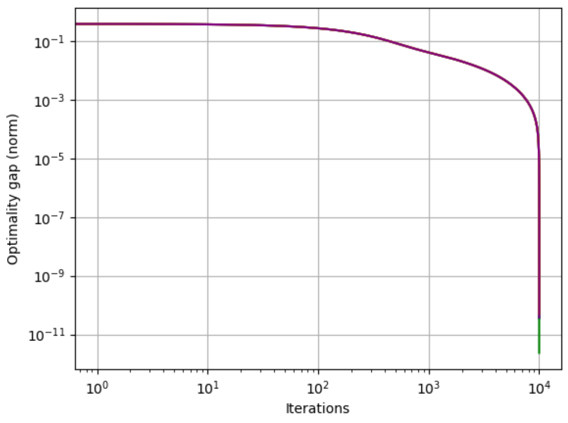


(c) DGD with Fully Connected Graph

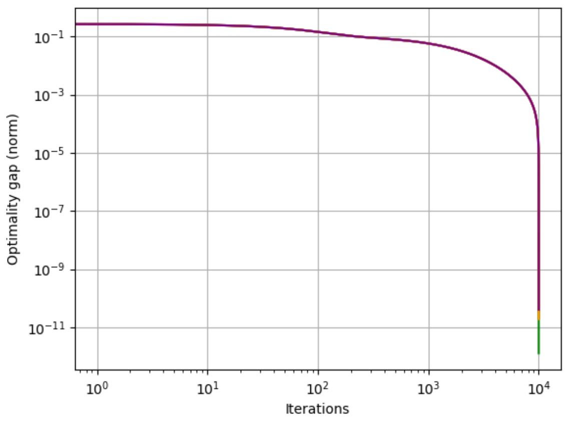
Figure 5: Convergence of DGD depending on the graph, stepsize = 0.002



(a) GT with Line Graph

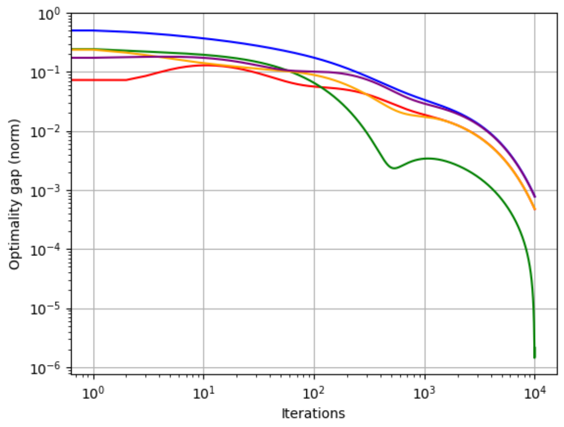


(b) GT with Small World Graph

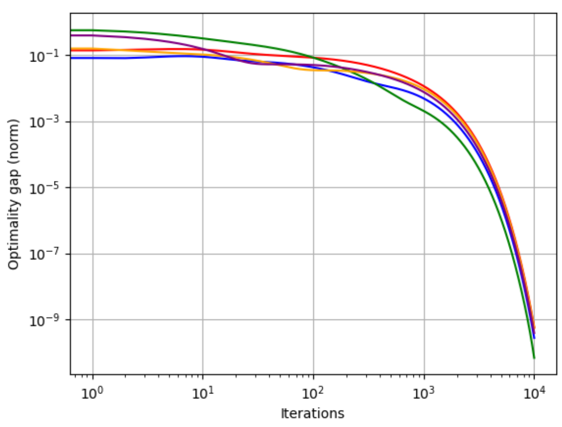


(c) GT with Fully Connected Graph

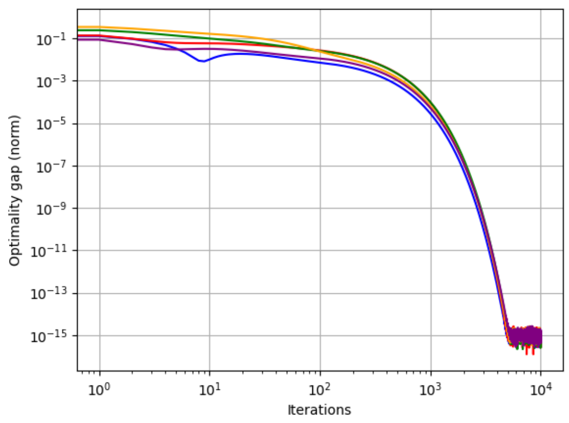
Figure 6: Convergence of GT depending on the graph, stepsize = 0.002



(a) Dual Decomposition with Line Graph

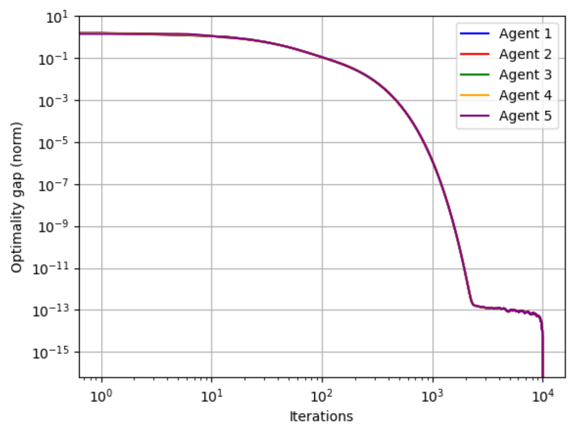


(b) Dual Decomposition with Small World Graph

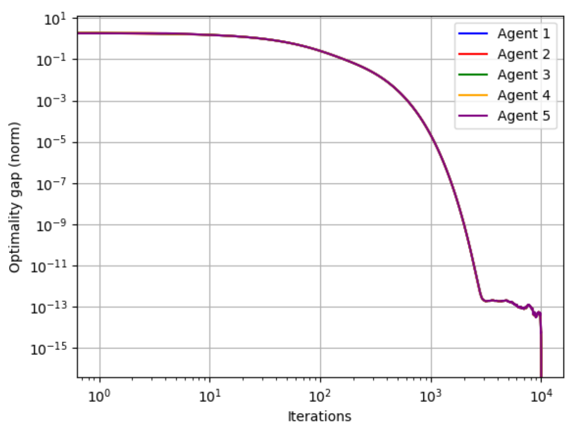


(c) Dual Decomposition with Fully Connected Graph

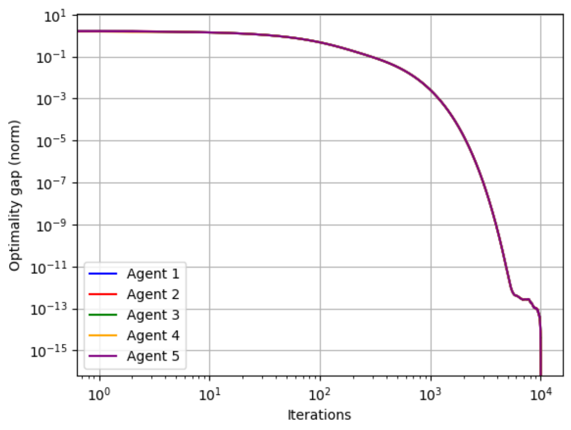
Figure 7: Convergence of Dual Decomposition depending on the graph, stepsize = 0.01



(a) ADMM with Line Graph



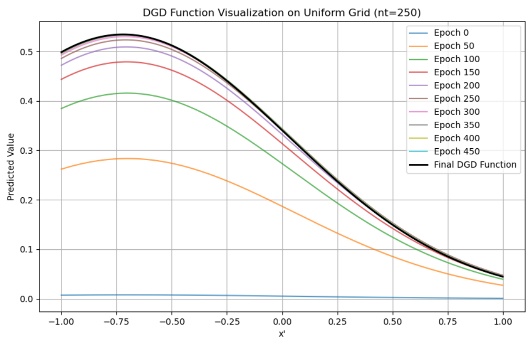
(b) ADMM with Small World Graph



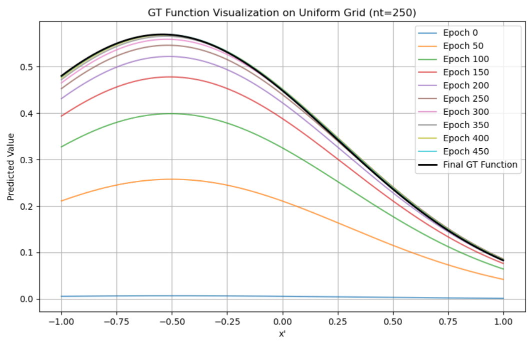
(c) ADMM with Fully Connected Graph

Figure 8: Convergence of ADMM depending on the graph, $\beta = 10$

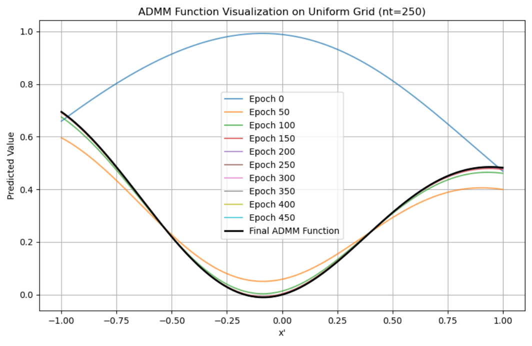
1.6 Uniform grid $nt = 250$



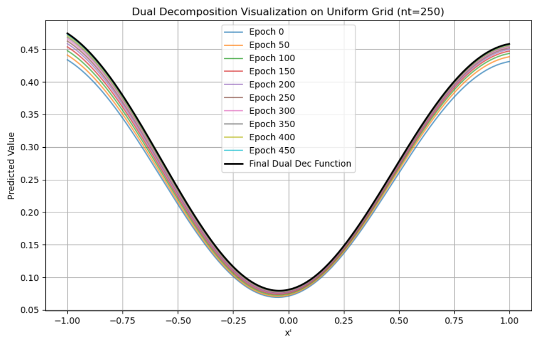
(a) DGD



(b) GT



(c) ADMM



(d) Dual Decomposition

Figure 9: Functions vizualisation on a uniform grid $nt = 250$

1.7 Convergence

1.8 Push-Sum Protocol

We implemented the Push-Sum protocol in order to recover convergence:

Push-sum Protocol:
At time k :

- Device j wakes up and sends two quantities to all its active neighbors: $\boldsymbol{x}_k^j / \left(1 + d^j_{\text{out },k}\right) \in \mathbf{R}^N$ and supporting variable $\varphi_k^j / \left(1 + d^j_{\text{out},k}\right) \in \mathbf{R}$. Here d^j_{out} is the outer degree: the number of neighbors device j communicates to.
- Each device i computes,

$$\boldsymbol{x}_{k+1}^i = \sum_{j \in N_i^{\text{in}} \cup i} \frac{\boldsymbol{x}_k^j}{1 + d^j_{\text{out},k}}, \quad \varphi_{k+1}^i = \sum_{j \in N_i^{\text{in}} \cup i} \frac{\varphi_k^j}{1 + d^j_{\text{out},k}}, \quad z_{k+1}^i = \frac{\boldsymbol{x}_{k+1}^i}{\varphi_{k+1}^i},$$

here N_i^{in} is the set of neighbors that are communicating to i (incoming).

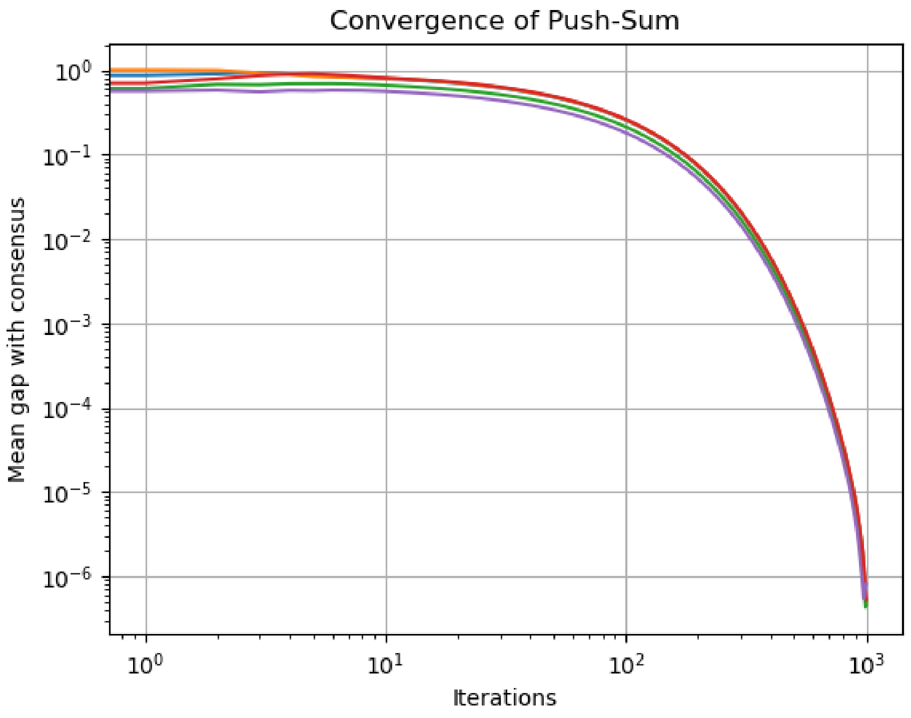


Figure 10: Push-Sum convergence

The directed graph is given by a function `directed_graph(a, density=0.5)` and here is the adjacency matrix:

```
Adjacency matrix W:
 [[0. 1. 0. 0. 0.]
 [1. 0. 1. 0. 1.]
 [0. 1. 0. 1. 0.]
 [0. 1. 0. 0. 1.]
 [1. 0. 0. 0. 0.]]
```

1.9 Different values of n

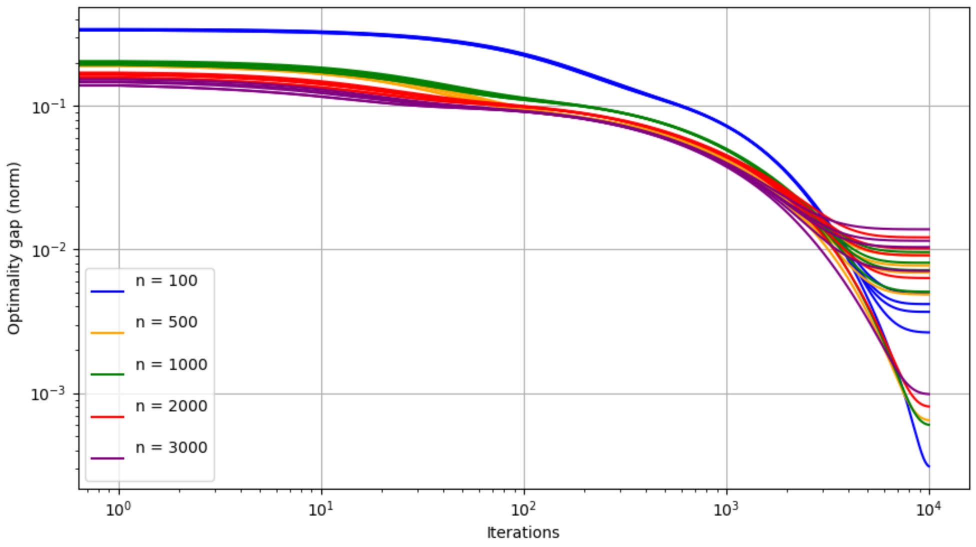


Figure 11: DGD for different values of n

1.9.1 Convergence Analysis with Respect to Sample Size n

The convergence behavior of the algorithms — namely DGD, GT, ADMM, and Dual Decomposition — is influenced by the number of samples n due to increased computational and communication complexity. We couldn't test higher n values for ADMM, GT and Dual Decomposition due to computational costs and technical problems [for $n = 10000$ the code didn't finish even after 6hours], but we believe that their behavior is no far than the DGD

As the samplis due to **Increased Problem Size:**, Larger n implies handling bigger kernel matrices, making linear system solutions more computationally expensive. Also The conditioning of matrices K_{mm} and K_{nm} worsens, slowing convergence and larger n leads to higher communication costs since every agent handles a very large sample of data points

- **DGD:** The convergence rate typically scales as $O(1/t)$, where t denotes the iteration count. As n increases, the gradient computation per agent becomes more expensive, leading to slower convergence.

2 Part II (Classes 3 and 4)

2.1 FedAvg

In this section, we revisit the problem with parameters $n = 100$, $m = 10$, and $a = 5$, introducing a second dataset containing $X[i]$ and $Y[i]$ for each agent i. We employ Federated Learning to address this scenario, reformulating the optimization problem as:

$$\alpha^* = \arg\min_{\alpha\in\mathbb{R}^m} \sum_{i=1}^{5}\sum_{j=1}^{B}\left[\frac{1}{B}\frac{1}{5}\frac{\sigma^2}{2}\alpha^T K_{mm}\alpha + \frac{1}{2}\sum_{l\in B_j}|y_l - K_{(l)m}\alpha|^2\right] \tag{1}$$

Let us define the function g_j for each agent as:

$$g_j(\alpha) = \frac{1}{B}\frac{1}{5}\frac{\sigma^2}{2}\alpha^T K_{mm}\alpha + \frac{1}{2}\sum_{l\in B_j} |y_l - K_{(l)m}\alpha|^2 \tag{2}$$

Subsequently, we implement FedAvg with the following parameters: $B = 20$ batches, $E = 10$ epochs, and $C = 5$ clients selected at each epoch.

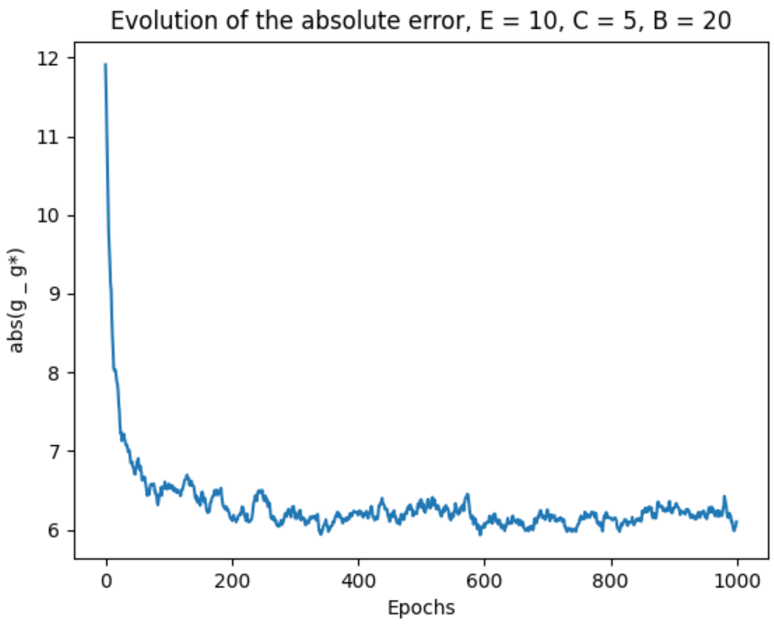


Figure 12: Absolute error in objective for FedAvg algorithm

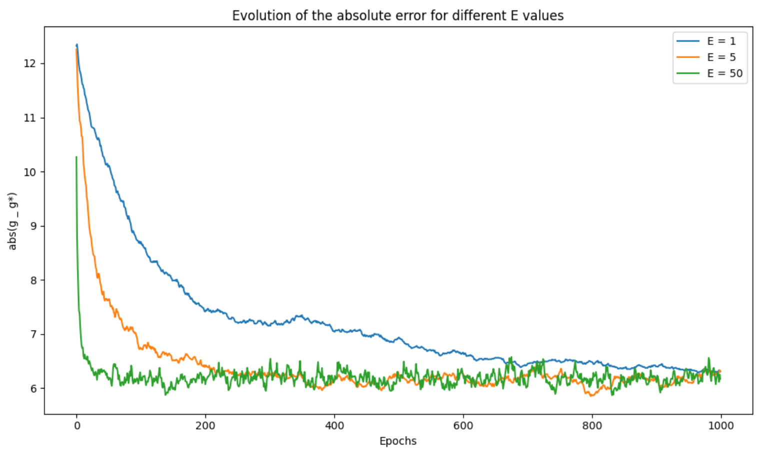
The absolute error $(g(\alpha_{optim}) - g^*)$ in figure 12 is not decreasing strictly (stays arround 6.5 for 800 epochs) and increases from time to time. We think this is due to the local stochastic gradient algorithm. The convergence is not as good as with DGD but still pretty good. This is beacause Federated learning prior goal is to improve individual client models through collaborative solution sharing across the network, rather than looking to optimal centralized solution

2.2 FedAvg with different parameters

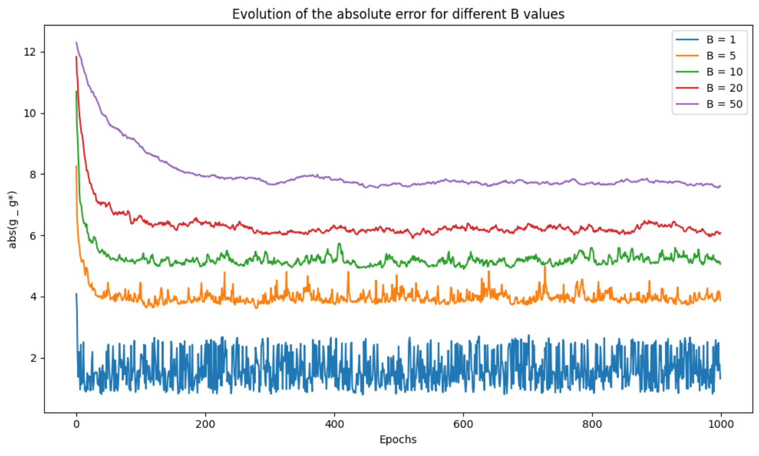
Now we test the FedAvg algorithm with different parameters (E, B, C).

Th results in figure 13 show the effects of each parameter on the performance :

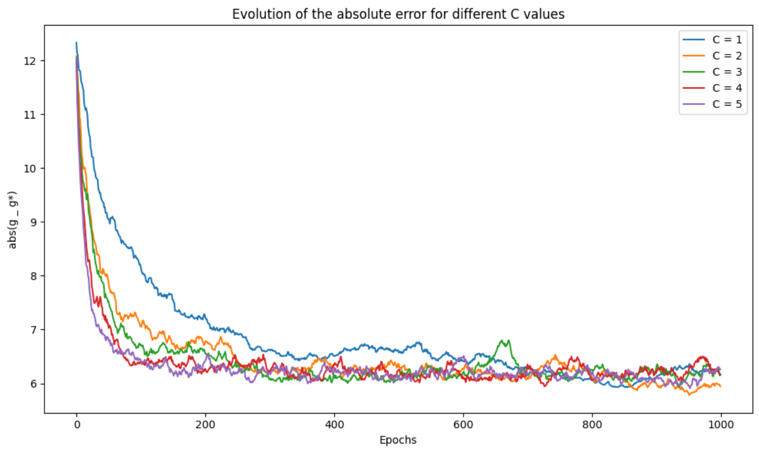
- **Number of local epochs** : Increasing the number of epochs in the local SGD leads to more variations in the curve (more error). But we also see that adding more epochs leads to faster convergence (We reach the minimum of 6.5 faster) so it is the classical tradeoff between speed of convergence and precision. We also see that the three curves reached the same minimum. So adding more epochs doesn't improve performance.
- **Number of batches** : We can see that high number of batches leads to stable but slow convergence, in the other hand, with B = 1 (taking all the data everytime) leads to faster convergence but with more error variations. Also, higher number of batches have higher absolute error $(g(\alpha_{optim}) - g^*)$.



(a) Varying number of local epochs E



(b) Varying number of batches B



(c) Varying number of clients selected C

Figure 13: Absolute error of FedAvg algorithm with diferent hyperparameters

- **Number of clients selected** : Increasing the number of clients leads to faster convergence, but all the curves reach the same minimum in the end

3 Part III (Class 5)

3.1 Experimental Setup

For our experiments, we used the following parameters:

- Noise variance: $\sigma^2 = 0.25$
- Regularization parameter: $\nu = 1$
- Smoothness parameter: $\beta = 10$
- Number of epochs: $n\_epochs = 10000$
- Standard deviation: $\sigma = 0.5$
- Privacy parameters: $\epsilon \in 0.1, 1, 10$

We tested the algorithm in two different network settings:

1. Small network: $n = 100$ data points, $m = 10$ features, $a = 5$ agents
2. Large network: $n = 1000$ data points, $m = 33$ features, $a = 100$ agents

3.2 Results and Analysis

3.2.1 Small Network Setting

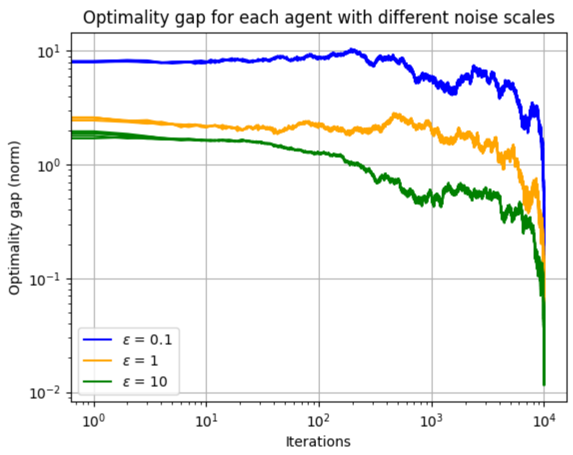


Figure 14: Optimality gap of DP-DGD with $n = 100$, $m = 10$, $a = 5$ for different noises

Our experimental results clearly demonstrate that increasing the privacy parameter ϵ leads to better optimization performance across both network settings. As shown in the figures, the green lines ($\epsilon = 10$) consistently achieve lower optimality gaps compared to the orange ($\epsilon = 1$) and blue ($\epsilon = 0.1$) lines.

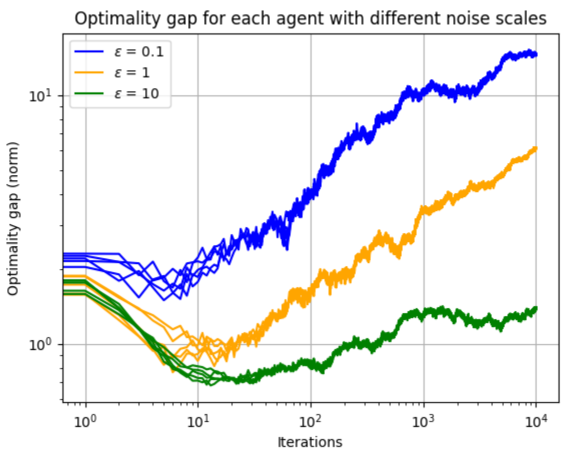


Figure 15: Optimality gap of DP-DGD with $n = 100$, $m = 10$, $a = 5$ for different noises

3.2.2 Large Network Setting

When scaling to the larger network with 100 agents (and more data poits, we observe that the algorithm initially starts to converge (optimality gaps gradually decreasing) and this convergent pattern stays until approximately the 100th iteration, where a sudden and dramatic divergence occurs for all privacy levels.

4 Conclusion

In this project, we investigated various distributed optimization algorithms: DGD, Gradient Tracking, Dual Decomposition, ADMM, FedAvg, and DGD-DP. A significant challenge was that our kernel ridge regression problem lacked strong convexity, preventing direct application of some theoretical convergence guarantees from the course.

Gradient Tracking and DGD exhibited similar convergence patterns, with GT showing more consistent behavior across agents. Despite theoretical assumptions not being fully satisfied for ADMM and Dual Decomposition (particularly regarding the Hessian eigenvalue conditions), both algorithms successfully converged to accurate solutions.

The network topology significantly impacted convergence, with fully connected graphs generally outperforming line and small-world graphs across all algorithms. For federated learning, FedAvg achieved satisfactory convergence, though not matching DGD's performance. Our parameter studies showed that increasing local epochs (E) accelerated convergence at the cost of stability, while larger batch counts (B) provided smoother but slower convergence.

In the differential privacy experiments, we observed the fundamental privacy-utility trade-off, with stronger privacy guarantees (smaller ϵ) resulting in larger optimality gaps, particularly in smaller networks. Computational efficiency remained reasonable, with all Part I algorithms converging within minutes for 100,000 iterations.

Future work could explore the effects of directed communication, packet loss, and asynchronous updates, which we were unable to investigate due to time constraints spent resolving implementation challenges.