INSTITUT POLYTECHNIQUE DE PARIS

ENSTA IP PARIS

SOD314. Cooperative Optimization for data science

Andrea Simonetto

Academic year 2022/2023

General info

- 5 classes (2h class, 1h30 project)
- All the info are on moodle
- 2h written Exam: 28/03, with material
- Grade: a function of Exam, TP, and optional homework along the way (it's the average between exam and TP plus extra points due to homework, but the result is cropped so that it cannot be more or less than 3 points of the exam grade)
- Extra points due to homework: max 3 points in total
- Contacts
  Andrea SIMONETTO
  andrea.simonetto@ensta-paris.fr,

Content

1. Introduction
2. Distributed Optimization
3. Intermezzo: Stochastic gradient
4. Federated Learning
5. Differential Privacy

Material: Slides, and suggested references to books/articles as we go along.

Part I
Introduction

Some recap of useful notions

- We look at continuous optimization problems (like in OPT201, OPT202), i.e.,
$$\min_{x \in X \subseteq \mathrm{R}^n} f(x), \tag{1}$$
where the set X is closed and convex, $f : \mathbf{R}^n \to \mathbf{R}$ is a convex function and the minimum is attained for a $x \in X$.

Some recap of useful notions

- We look at continuous optimization problems (like in OPT201, OPT202), i.e.,

$$\min_{x \in X \subseteq \mathrm{R}^n} f(x), \tag{1}$$

  where the set X is closed and convex, $f : \mathbf{R}^n \to \mathbf{R}$ is a convex function and the minimum is attained for a $x \in X$.
- What you need to remember from past courses will be reviewed as we go along, but mainly:
  - Optimality conditions (KKT), which are here necessary and sufficient (under a Slater's constraint qualification assumption)
  - How to derive dual problems (Lagrangian function, dual function, problems)
  - Subgradients
  - Algorithms: How to derive first-order algorithms (e.g., gradient descent) from the optimality conditions, and how to prove their convergence and convergence rate
  - Some basic linear algebra: eigenvalues, singular value decomposition, solution of linear systems, etc..

The optimization problem of interest

- Said so, the problem we will look at in this course has the form,

$$\text{(P)} \qquad \min_{x \in X \subseteq \mathbb{R}^n} \sum_{i=1}^{N} f_i(x), \tag{2}$$

where the set X is closed and convex, $f_i : \mathbf{R}^n \to \mathbf{R}$ is a convex function for all i's and the minimum is attained for a $x \in X$.
Here N are the number of agents/players/nodes/sub-systems/etc.. that cooperate to solve the optimization problem.

Examples and applications

- Cooperative least-squares. Imagine N users collect noisy measurements $\boldsymbol{y}_i \in \mathbf{R}^n$ about a quantity $\boldsymbol{x} \in X$. A way to estimate the true value for $\boldsymbol{x}$ is to set up a least-squares problem as

$$\min_{x \in X} \sum_{i=1}^{N} \|\boldsymbol{x} - \boldsymbol{y}_i\|^2 = \sum_{i=1}^{N} f_i(\boldsymbol{x})$$

Examples and applications

- Cooperative least-squares. Imagine N users collect noisy measurements $\boldsymbol{y}_i \in \mathbf{R}^n$ about a quantity $\boldsymbol{x} \in X$. A way to estimate the true value for $\boldsymbol{x}$ is to set up a least-squares problem as

$$\min_{\boldsymbol{x} \in X} \sum_{i=1}^{N} \|\boldsymbol{x} - \boldsymbol{y}_i\|^2 = \sum_{i=1}^{N} f_i(\boldsymbol{x})$$

- Cooperative linear model training. Imagine N users collect input-output pairs $\boldsymbol{w}_i \in \mathbf{R}^{n-1}, \boldsymbol{y}_i \in \mathbf{R}$ (e.g., features and labels), and they want to train a global model with weights $\boldsymbol{x} \in X \subseteq \mathbf{R}^n, \boldsymbol{x} = [\theta \in \mathbf{R}^{n-1}, c \in \mathbf{R}]$. Let the local input-output mapping being affine,

$$\boldsymbol{y}_i = \theta^\top \boldsymbol{w}_i + c, \qquad \forall i,$$

  then the training problem can be written as

$$\min_{\boldsymbol{x} \in X} \sum_{i=1}^{N} \|\boldsymbol{y}_i - (\theta^\top \boldsymbol{w}_i + c)\|^2 = \sum_{i=1}^{N} f_i(\boldsymbol{x})$$

Examples and applications

- Cooperative network problems. Take a sensor network of N sensors. you want to compute the localization of the whole network based on pair-wise distance measurements (e.g., sensors can be cars, or people with their phones). Then each measurement is

$$m_{ij} = \|x_i - x_j\| + \text{noise}, \qquad x_i \in \mathbf{R}^2 \text{ is the position of node } i.$$

  You have E pair-wise measurements. Then you can write the problem as

$$\min_{x \in X \subset \mathrm{R}^{2E}} \sum_{i,j}^{E} (m_{ij} - \|x_i - x_j\|)^2 = \sum_{k=1}^{E} f_k(x)$$

Examples and applications

- Cooperative network problems. Take a sensor network of N sensors. you want to compute the localization of the whole network based on pair-wise distance measurements (e.g., sensors can be cars, or people with their phones). Then each measurement is

$$m_{ij} = \|x_i - x_j\| + \text{noise}, \qquad x_i \in \mathbf{R}^2 \text{ is the position of node } i.$$

  You have E pair-wise measurements. Then you can write the problem as

$$\min_{x \in X \subset \mathrm{R}^{2E}} \sum_{i,j}^{E} (m_{ij} - \|x_i - x_j\|)^2 = \sum_{k=1}^{E} f_k(x)$$

- Cooperative consensus. You have N robots at different locations $\boldsymbol{s}_i$ and moving at different speeds v_i, and you want to find the best position in space for the fastest rendez-vous:

$$\min_{x \in X \subset \mathrm{R}^3} \sum_{i}^{N} \frac{\|\boldsymbol{s}_i - x\|}{v_i} = \sum_{i=1}^{N} f_i(x)$$

Main challenges and plan for the course

- Large-scale problems: N is big or n is big and the problem cannot be solved (not stored) locally. You need to solve it on separate machines, or iteratively.
  - If n is big, usually we talk about parallel methods;
  - If N is big, usually we talk about distributed methods.

Main challenges and plan for the course

- Large-scale problems: N is big or n is big and the problem cannot be solved (not stored) locally. You need to solve it on separate machines, or iteratively.
  - If n is big, usually we talk about parallel methods;
  - If N is big, usually we talk about distributed methods.
- Distributed localization: the data that builds f_i is stored in separate machines (think: mobile phones). Then, you have communication issues and latencies..
  - Are we communicating to a server? (Cloud-based)
  - Are we communicating to each other (Peer-to-peer)
  - Common issues: asynchronicity, bi-directionality, packet-losses, communication overhead,..
  - Here: graph theory to the rescue!

The SneakerNet paradox. What is the fastest way to send large data sets?

Main challenges and plan for the course

- Large-scale problems: N is big or n is big and the problem cannot be solved (not stored) locally. You need to solve it on separate machines, or iteratively.
  - If n is big, usually we talk about parallel methods;
  - If N is big, usually we talk about distributed methods.
- Distributed localization: the data that builds f_i is stored in separate machines (think: mobile phones). Then, you have communication issues and latencies..
  - Are we communicating to a server? (Cloud-based)
  - Are we communicating to each other (Peer-to-peer)
  - Common issues: asynchronicity, bi-directionality, packet-losses, communication overhead,..
  - Here: graph theory to the rescue!

The SneakerNet paradox. What is the fastest way to send large data sets? FedEx

Main challenges and plan for the course

- Large-scale problems: N is big or n is big and the problem cannot be solved (not stored) locally. You need to solve it on separate machines, or iteratively.
  - If n is big, usually we talk about parallel methods;
  - If N is big, usually we talk about distributed methods.
- Distributed localization: the data that builds f_i is stored in separate machines (think: mobile phones). Then, you have communication issues and latencies..
  - Are we communicating to a server? (Cloud-based)
  - Are we communicating to each other (Peer-to-peer)
  - Common issues: asynchronicity, bi-directionality, packet-losses, communication overhead,..
  - Here: graph theory to the rescue!
- Data and functions are private: you don't want to disclose f_i or the local decision to the other players. How do you do that? (Example: training language models based on private text messages on your phone)

Main challenges and plan for the course

- In this course, we will look at algorithms to tackle the challenges above while solving problem (P)

Main challenges and plan for the course

- In this course, we will look at algorithms to tackle the challenges above while solving problem (P)
- The algorithms will be of the first-order kind (more later)

Main challenges and plan for the course

- In this course, we will look at algorithms to tackle the challenges above while solving problem (P)
- The algorithms will be of the first-order kind (more later)
- We divide the course in three parts (distributed, federated, private)

Part II
Basics

Some definitions

Definition 1 (Convex functions)

A function $f : X \subseteq \mathbf{R}^n \to \mathbf{R}$ is convex iff X is convex and

$$\text{(C1)} \quad \forall \boldsymbol{x}, \boldsymbol{y} \in X, \lambda \in [0, 1] : \quad f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}).$$

Multiple definitions exists, for example:

$$\text{(C1)} + f \in \mathcal{C}^1(X) \quad \Longleftrightarrow \quad \forall \boldsymbol{x}, \boldsymbol{y} \in X,\ f(\boldsymbol{x}) \geq f(\boldsymbol{y}) + \langle \nabla f(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \rangle \tag{3}$$

$$\text{(C1)} + f \in \mathcal{C}^2(X) \quad \Longleftrightarrow \quad \forall \boldsymbol{x}, \boldsymbol{y} \in X,\ \nabla^2 f(\boldsymbol{x}) \succeq 0 \tag{4}$$

Proof. Homework.

Some definitions

Definition 1 (Strongly convex functions)

A convex function $f : X \subseteq \mathbf{R}^n \to \mathbf{R}$ is *m*-strongly convex iff

$$(\mathrm{SC}) \quad f(\mathbf{x}) - \frac{m}{2}\|\mathbf{x}\|^2 \text{ is convex.}$$

f doesn't need to be differentiable!
Multiple definitions exists, for example:

$$(\mathrm{SC}) + f \in \mathcal{C}^1(X) \iff \forall \mathbf{x}, \mathbf{y} \in X,\ \langle \nabla f(\mathbf{x}) - \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle \geq m\|\mathbf{x} - \mathbf{y}\|^2 \quad (3)$$

$$(\mathrm{SC}) + f \in \mathcal{C}^2(X) \iff \forall \mathbf{x}, \mathbf{y} \in X,\ \nabla^2 f(\mathbf{x}) \succeq m I_n \quad (4)$$

Proof. Homework.

Some definitions

Definition 1 (Smooth functions)

A convex function $f : X \subseteq \mathbf{R}^n \to \mathbf{R}$ is L-smooth iff

$$\text{(LC)} \quad \frac{L}{2}\|x\|^2 - f(x) \text{ is convex.}$$

Important (LC) $\implies f \in \mathcal{C}^1(X)$!
Multiple definitions exists, for example:

$$\text{(LC)} \iff \forall x, y \in X,\ \|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\| \tag{3}$$

$$\text{(LC)} \iff \forall x, y \in X,\ \frac{1}{L}\|\nabla f(x) - \nabla f(y)\|^2 \leq \langle \nabla f(x) - \nabla f(y), x - y\rangle \tag{4}$$

$$\text{(LC)} + f \in \mathcal{C}^2(X) \iff \forall x, y \in X,\ 0 \preceq \nabla^2 f(x) \preceq L I_n \tag{5}$$

Proof. Homework.

Gradient algorithm

Consider solving $\min_{x \in \mathbb{R}^n} f(x)$.
Rem: the gradient is an ascent direction, and a first-order method

Gradient algorithm

Consider solving $\min_{x \in \mathbb{R}^n} f(x)$.
Rem: the gradient is an ascent direction, and a first-order method
The simplest scheme, let $\alpha_k > 0$:

- Start with $x_0 \in \mathbf{R}^n$
- Iterate $x_{k+1} = x_k - \alpha_k \nabla f(x_k), \quad k = 0, 1, \ldots.$

There are different methods to choose α_k (either a priori or online):

- Constant: $\alpha_k = \alpha$
- Vanishing: $\alpha_k = \frac{\alpha}{\sqrt{k+1}}$
- ...

Why the difference? Convergence

Gradient algorithm: convergence recap

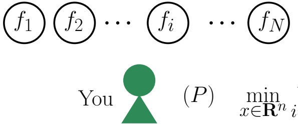
- Recap from OPT201-202: gradient converges in a well-defined sense provided the step size α_k is chosen appropriately.
- Depending on the functional class, the gradient algorithm behaves differently with the number of iterations t.
- The table below recaps the main results that one can expect:

Type	just convex	smooth	smooth+strongly convex
	First-order		
Convergence guarantee	$\|\nabla f\|, f - f^*$	$f - f^*$	$\|x - x^*\|$
Convex	$O(1/\sqrt{t})$, $O(1/t)$	$O(1/t)$, $O(1/t^2)$	$O(\rho^t)$

Homework: Revise your theory

Let's finally start!

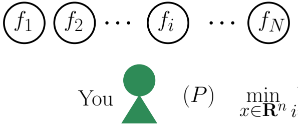
- We want to solve problem (P) but N is too big. How do we do?



$$(P) \quad \min_{x \in \mathbf{R}^n} \sum_{i=1}^{N} f_i(x)$$

Let's finally start!

- We want to solve problem (P) but N is too big. How do we do?



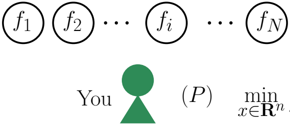
$$(P) \quad \min_{x\in\mathbf{R}^n} \sum_{i=1}^{N} f_i(x)$$

- First possibility: you ask one device to apply a local gradient, then you get back the result and you ask a second device..

Incremental/ Gauss-Seidel Gradient

Let's finally start!

- We want to solve problem (P) but N is too big. How do we do?



$$(P) \quad \min_{x \in \mathbf{R}^n} \sum_{i=1}^{N} f_i(x)$$

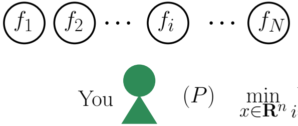
- First possibility: you ask one device to apply a local gradient, then you get back the result and you ask a second device..

Incremental/ Gauss-Seidel Gradient

  - Start with $x_0 \in \mathbf{R}^n$

Let's finally start!

- We want to solve problem (P) but N is too big. How do we do?



$$(P) \quad \min_{x \in \mathbf{R}^n} \sum_{i=1}^{N} f_i(x)$$

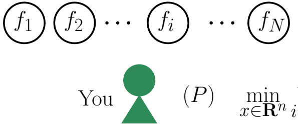
- First possibility: you ask one device to apply a local gradient, then you get back the result and you ask a second device..

Incremental/ Gauss-Seidel Gradient

  - Start with $x_0 \in \mathbf{R}^n$
  - Iterate: ask device i: $x_{k+1} = x_k - \alpha_k \nabla f_i(x_k), \quad k = 0, 1, \ldots.$

Let's finally start!

- We want to solve problem (P) but N is too big. How do we do?



$$(P) \quad \min_{x \in \mathbf{R}^n} \sum_{i=1}^{N} f_i(x)$$

- First possibility: you ask one device to apply a local gradient, then you get back the result and you ask a second device..

Incremental/ Gauss-Seidel Gradient

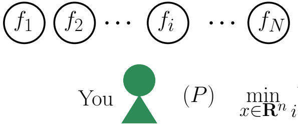
  - Start with $x_0 \in \mathbf{R}^n$
  - Iterate: ask device i: $x_{k+1} = x_k - \alpha_k \nabla f_i(x_k), \quad k = 0, 1, \ldots.$

- Second possibility: you ask every device to apply a local gradient, then you get back the result and ??

Parallel/ Jacobi Gradient

Let's finally start!

- We want to solve problem (P) but N is too big. How do we do?



$$(P) \quad \min_{x \in \mathbf{R}^n} \sum_{i=1}^{N} f_i(x)$$

- First possibility: you ask one device to apply a local gradient, then you get back the result and you ask a second device..

Incremental/ Gauss-Seidel Gradient

  - Start with $x_0 \in \mathbf{R}^n$
  - Iterate: ask device i: $x_{k+1} = x_k - \alpha_k \nabla f_i(x_k), \quad k = 0, 1, \ldots.$

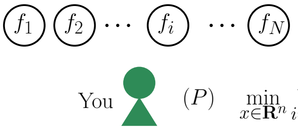
- Second possibility: you ask every device to apply a local gradient, then you get back the result and ??

Parallel/ Jacobi Gradient

  - Start with $x_0 \in \mathbf{R}^n$

Let's finally start!

- We want to solve problem (P) but N is too big. How do we do?



$$(P) \quad \min_{x \in \mathbf{R}^n} \sum_{i=1}^{N} f_i(x)$$

- First possibility: you ask one device to apply a local gradient, then you get back the result and you ask a second device..

Incremental/ Gauss-Seidel Gradient

  - Start with $x_0 \in \mathbf{R}^n$
  - Iterate: ask device i: $x_{k+1} = x_k - \alpha_k \nabla f_i(x_k), \quad k = 0, 1, \ldots.$

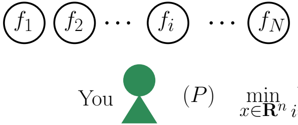
- Second possibility: you ask every device to apply a local gradient, then you get back the result and ??

Parallel/ Jacobi Gradient

  - Start with $x_0 \in \mathbf{R}^n$
  - Iterate: ask every devices: $x^i_{k+1} = x_k - \alpha_k \nabla f_i(x_k), \quad \forall i$

Let's finally start!

- We want to solve problem (P) but N is too big. How do we do?



$$(P) \quad \min_{x \in \mathbf{R}^n} \sum_{i=1}^{N} f_i(x)$$

- First possibility: you ask one device to apply a local gradient, then you get back the result and you ask a second device..

Incremental/ Gauss-Seidel Gradient

  - Start with $x_0 \in \mathbf{R}^n$
  - Iterate: ask device i: $x_{k+1} = x_k - \alpha_k \nabla f_i(x_k), \quad k = 0, 1, \ldots.$

- Second possibility: you ask every device to apply a local gradient, then you get back the result and ??

Parallel/ Jacobi Gradient

  - Start with $x_0 \in \mathbf{R}^n$
  - Iterate: ask every devices: $x^i_{k+1} = x_k - \alpha_k \nabla f_i(x_k), \quad \forall i$
  - $x_{k+1} = ?$

Properties and convergence of incremental gradient

- The data that generates f_i stays private, and if you have enough time you don't care about latencies (etc..): the scheme is quite robust

Properties and convergence of incremental gradient

- The data that generates f_i stays private, and if you have enough time you don't care about latencies (etc..): the scheme is quite robust
- Also known in the ML community as: online back-propagation, finite sum stochastic gradient descent (careful: step size is called the learning rate)

Theorem 2

Consider problem (P) for a m-strongly convex and L-smooth function f. Consider the incremental gradient method with step size α_k. Assume that $\|\nabla f_i(x) - \sum_i \nabla f_i(x)\| \leq G$ for all $x \in \mathbf{R}^n$, choose $\alpha_k < 2/L$, and define the quantity, $\rho_k = \max\{|1-\alpha_k m|, |1-\alpha_k L|\} < 1$.
Then convergence of the incremental gradient goes as

$$\|x_{k+1} - x^*\| \leq \rho_k \|x_k - x^*\| + \alpha_k G.$$

Corollary 3

If $\alpha_k = 1/k^s$, $0 < s < 1$ then, $\|x_k - x^*\| \leq O(1/k^s)$.

Proof

- Define $f(\mathbf{x}) = \sum_i f_i(\mathbf{x})$. Write

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \|\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k) - \mathbf{x}^* + \alpha_k \nabla f(\mathbf{x}^*)\| + \alpha_k \|\nabla f_i(\mathbf{x}_k) - \nabla f(\mathbf{x}_k)\|$$

Proof

- Define $f(\mathbf{x}) = \sum_i f_i(\mathbf{x})$. Write

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \|\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k) - \mathbf{x}^* + \alpha_k \nabla f(\mathbf{x}^*)\| + \alpha_k \|\nabla f_i(\mathbf{x}_k) - \nabla f(\mathbf{x}_k)\|$$

- Use strongly convex and smoothness property to say

$$\|\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k) - \mathbf{x}^* + \alpha_k \nabla f(\mathbf{x}^*)\| \leq \rho_k \|\mathbf{x}_k - \mathbf{x}^*\|$$

Proof

- Define $f(x) = \sum_i f_i(x)$. Write

$$\|x_{k+1} - x^*\| \leq \|x_k - \alpha_k \nabla f(x_k) - x^* + \alpha_k \nabla f(x^*)\| + \alpha_k \|\nabla f_i(x_k) - \nabla f(x_k)\|$$

- Use strongly convex and smoothness property to say

$$\|x_k - \alpha_k \nabla f(x_k) - x^* + \alpha_k \nabla f(x^*)\| \leq \rho_k \|x_k - x^*\|$$

- The thesis follows. The corollary is a bit more involved but not impossible, see for example: arXiv:1510.08562

Proof

- Define $f(x) = \sum_i f_i(x)$. Write

$$\|x_{k+1} - x^*\| \leq \|x_k - \alpha_k \nabla f(x_k) - x^* + \alpha_k \nabla f(x^*)\| + \alpha_k \|\nabla f_i(x_k) - \nabla f(x_k)\|$$

- Use strongly convex and smoothness property to say

$$\|x_k - \alpha_k \nabla f(x_k) - x^* + \alpha_k \nabla f(x^*)\| \leq \rho_k \|x_k - x^*\|$$

- The thesis follows. The corollary is a bit more involved but not impossible, see for example: arXiv:1510.08562

- A lot of research is devoted in lifting the assumptions (strong convexity, smoothness), and relaxing the assumption on G, but the basic ideas are still valid: you have a trade-off between speed and error.

Convergence of parallel gradient?

- It seems that it could be a faster algorithm, but how can we combine different updates x^i_{k+1} ? Can we just average them ?

Convergence of parallel gradient?

- It seems that it could be a faster algorithm, but how can we combine different updates x^i_{k+1} ? Can we just average them ?
- The scheme is less robust to asynchronicity, package drops, etc..

Convergence of parallel gradient?

- It seems that it could be a faster algorithm, but how can we combine different updates x^i_{k+1} ? Can we just average them ?
- The scheme is less robust to asynchronicity, package drops, etc..
- The scheme is the starting point of distributed optimization (next!)

Part III
Distributed optimization

Let's go Graph theory!

- We will need some basic graph theory to devise methods to combine different updates in the parallel scheme and analyze communication issues

Let's go Graph theory!

- We will need some basic graph theory to devise methods to combine different updates in the parallel scheme and analyze communication issues
- Consider a graph $\mathcal{G}$ as a collection of vertices $\mathcal{V}$, edges $\mathcal{E}$, and edge weights $\mathcal{W}$.

Let's go Graph theory!

- We will need some basic graph theory to devise methods to combine different updates in the parallel scheme and analyze communication issues
- Consider a graph $\mathcal{G}$ as a collection of vertices $\mathcal{V}$, edges $\mathcal{E}$, and edge weights $\mathcal{W}$.
- We define undirected a graph for which an edge can carry information in both directions. Otherwise we call it directed.

Let's go Graph theory!

- We will need some basic graph theory to devise methods to combine different updates in the parallel scheme and analyze communication issues
- Consider a graph $\mathcal{G}$ as a collection of vertices $\mathcal{V}$, edges $\mathcal{E}$, and edge weights $\mathcal{W}$.
- We define undirected a graph for which an edge can carry information in both directions. Otherwise we call it directed.
- We define the Adjacency matrix as $\mathcal{A} \in \{0, 1\}^{|\mathcal{V}|\times|\mathcal{V}|}$, with 1 entries if node i and j share an edge.

Let's go Graph theory!

- We will need some basic graph theory to devise methods to combine different updates in the parallel scheme and analyze communication issues
- Consider a graph $\mathcal{G}$ as a collection of vertices $\mathcal{V}$, edges $\mathcal{E}$, and edge weights $\mathcal{W}$.
- We define undirected a graph for which an edge can carry information in both directions. Otherwise we call it directed.
- We define the Adjacency matrix as $\mathcal{A} \in \{0, 1\}^{|\mathcal{V}|\times|\mathcal{V}|}$, with 1 entries if node i and j share an edge.
- The Laplacian matrix of a undirected graph $\mathcal{G}$ is an $|\mathcal{V}| \times |\mathcal{V}|$ symmetric matrix defined by

  $$\mathcal{L}_{\mathcal{G}} = \mathcal{D} - \mathcal{A},$$

  where $\mathcal{D}$ =diag$(d_1, \dots, d_n)$ is the degree matrix, which is the diagonal matrix formed from the vertex degrees.
  Note that $\mathcal{L}_{\mathcal{G}}\mathbf{1} = \mathbf{0}$.

Let's go Graph theory!

- We will need some basic graph theory to devise methods to combine different updates in the parallel scheme and analyze communication issues
- Consider a graph $\mathcal{G}$ as a collection of vertices $\mathcal{V}$, edges $\mathcal{E}$, and edge weights $\mathcal{W}$.
- We define undirected a graph for which an edge can carry information in both directions. Otherwise we call it directed.
- We define the Adjacency matrix as $\mathcal{A} \in \{0, 1\}^{|\mathcal{V}|\times|\mathcal{V}|}$, with 1 entries if node i and j share an edge.
- The Laplacian matrix of a undirected graph $\mathcal{G}$ is an $|\mathcal{V}| \times |\mathcal{V}|$ symmetric matrix defined by
$$\mathcal{L}_{\mathcal{G}} = \mathcal{D} - \mathcal{A},$$
where $\mathcal{D}$ =diag$(d_1, \dots, d_n)$ is the degree matrix, which is the diagonal matrix formed from the vertex degrees.
Note that $\mathcal{L}_{\mathcal{G}}\mathbf{1} = \mathbf{0}$.
- A doubly stochastic matrix W is a matrix for which, $W\mathbf{1} = \mathbf{1}$, and $\mathbf{1}^\top W = \mathbf{1}^\top$.

Decentralized Gradient Descent (DGD)

- We are ready to get back to our parallel update method. Consider the following update:

Decentralized Gradient Descent (DGD)

- We are ready to get back to our parallel update method. Consider the following update:

Decentralized Gradient Descent

  - Start with $x_0^i \in \mathbf{R}^n$ at each device i
  - Iterate: ask every devices: $x_{k+1}^i = \sum_{j=1}^N w_{ij} x_k^j - \alpha_k \nabla f_i(x_k^i), \quad \forall i$

Here w_{ij} are not zero only if device i and j are not connected.

Decentralized Gradient Descent (DGD)

- We are ready to get back to our parallel update method. Consider the following update:

  Decentralized Gradient Descent

  - Start with $\mathbf{x}_0^i \in \mathbf{R}^n$ at each device i
  - Iterate: ask every devices: $\mathbf{x}_{k+1}^i = \sum_{j=1}^N w_{ij}\mathbf{x}_k^j - \alpha_k \nabla f_i(\mathbf{x}_k^i), \quad \forall i$

  Here w_{ij} are not zero only if device i and j are not connected.

- Effectively we are giving a copy of $\mathbf{x}$ to each device and looking at the iterates $\mathbf{y} = [\mathbf{x}^1; \mathbf{x}^2; \dots; \mathbf{x}^N]$

$$\mathbf{y}_{k+1} = W\mathbf{y}_k - \alpha_k \nabla_{\mathbf{y}} F(\mathbf{y}_k), \qquad F(\mathbf{y}) := \sum_{i=1}^N f_i(\mathbf{x}^i),$$

  where W is assumed doubly stochastic and it is the mixing matrix.

Decentralized Gradient Descent (DGD)

- We are ready to get back to our parallel update method. Consider the following update:

  Decentralized Gradient Descent

  - Start with $x_0^i \in \mathbf{R}^n$ at each device i
  - Iterate: ask every devices: $x_{k+1}^i = \sum_{j=1}^N w_{ij} x_k^j - \alpha_k \nabla f_i(x_k^i), \quad \forall i$

  Here w_{ij} are not zero only if device i and j are not connected.
- Effectively we are giving a copy of x to each device and looking at the iterates $y = [x^1; x^2; \ldots; x^N]$

$$y_{k+1} = W y_k - \alpha_k \nabla_y F(y_k), \qquad F(y) := \sum_{i=1}^N f_i(x^i),$$

where W is assumed doubly stochastic and it is the mixing matrix.
- DGD is peer-to-peer:
  - Each device gets and sends x_k^j / x_k^i from/to the neighbors
  - Each device computes: $x_{k+1}^i = \sum_{j=1}^N w_{ij} x_k^j - \alpha_k \nabla f_i(x_k^i), \quad \forall i$

DGD revisited

- DGD is the first distributed algorithm that we see in this course and it does not need a server. Each device can compute its version of the optimal decision.

DGD revisited

- DGD is the first distributed algorithm that we see in this course and it does not need a server. Each device can compute its version of the optimal decision.
- Communication needs to happen synchronously and bi-directionally

DGD revisited

- DGD is the first distributed algorithm that we see in this course and it does not need a server. Each device can compute its version of the optimal decision.
- Communication needs to happen synchronously and bi-directionally
- Other versions exist with the mixing matrix being applied before/after the local gradient, but the idea stays the same

DGD revisited

- DGD is the first distributed algorithm that we see in this course and it does not need a server. Each device can compute its version of the optimal decision.
- Communication needs to happen synchronously and bi-directionally
- Other versions exist with the mixing matrix being applied before/after the local gradient, but the idea stays the same
- What can we say about convergence? For a constant step size α, it turns out that it is not so difficult, since

$$\boldsymbol{y}_{k+1} = W\boldsymbol{y}_k - \alpha\nabla_{\boldsymbol{y}}F(\boldsymbol{y}_k) = \boldsymbol{y}_k - \alpha\Big[\nabla_{\boldsymbol{y}}F(\boldsymbol{y}_k) - \frac{1}{\alpha}(W-I)\boldsymbol{y}_k\Big]$$

so it's the same as optimizing,

$$\min_{\boldsymbol{y}\in\mathrm{R}^{Nn}} \alpha F(\boldsymbol{y}) + \frac{1}{2}\boldsymbol{y}^\top(I-W)\boldsymbol{y}.$$

DGD revisited

- DGD is the first distributed algorithm that we see in this course and it does not need a server. Each device can compute its version of the optimal decision.
- Communication needs to happen synchronously and bi-directionally
- Other versions exist with the mixing matrix being applied before/after the local gradient, but the idea stays the same
- What can we say about convergence? For a constant step size α, it turns out that it is not so difficult, since

$$\mathbf{y}_{k+1} = W\mathbf{y}_k - \alpha\nabla_{\mathbf{y}} F(\mathbf{y}_k) = \mathbf{y}_k - \alpha\Big[\nabla_{\mathbf{y}} F(\mathbf{y}_k) - \frac{1}{\alpha}(W - I)\mathbf{y}_k\Big]$$

so it's the same as optimizing,

$$\min_{\mathbf{y}\in\mathrm{R}^{Nn}} \alpha F(\mathbf{y}) + \frac{1}{2}\mathbf{y}^\top (I - W)\mathbf{y}.$$

- For a general diminishing α_k sequence, it is a bit more complex but doable

DGD convergence

- We assume that the mixing matrix $W = [w_{ij}]$ is symmetric and doubly stochastic. The eigenvalues of W are real and sorted in a nonincreasing order
$$1 = \lambda_1(W) \geq \lambda_2(W) \geq \ldots \geq \lambda_N(W) \geq -1.$$
Let the second largest magnitude of the eigenvalues of W be denoted as
$$\gamma = \max\{|\lambda_2(W)|, |\lambda_N(W)|\}$$

DGD convergence

- We assume that the mixing matrix $W = [w_{ij}]$ is symmetric and doubly stochastic. The eigenvalues of W are real and sorted in a nonincreasing order
$$1 = \lambda_1(W) \geq \lambda_2(W) \geq \ldots \geq \lambda_N(W) \geq -1.$$
Let the second largest magnitude of the eigenvalues of W be denoted as
$$\gamma = \max\{|\lambda_2(W)|, |\lambda_N(W)|\}$$
- Some basic questions in the decentralized/distributed setting arise: (1) When does x_k^i converge? (2) Does it converge to x^* (3) If not does consensus (i.e., $x_k^i = x_k^j$) hold asymptotically? (4) How do the properties of f_i and the network affect convergence?

DGD convergence: statement

Theorem 4 (DGD convergence)

Consider Problem (P) *and its solution via a decentralized gradient descent algorithm with constant step size* α, *and doubly stochastic communication matrix* W *with* $\gamma < 1$.
Let convex functions f_i *be* L_i*-smooth and let* $L = \max_i\{L_i\}$. *Let the mean value be*

$$\bar{x}_k = \frac{1}{N}\sum_{i=1}^{N} x_k^i.$$

If α *is chosen small enough and in particular* $\leq O(1/L)$, *then*

1. *Consensus:*

$$\|x_k^i - \bar{x}_k\| \to O(\frac{\alpha}{1-\gamma});$$

2. *Convergence*

$$f(\bar{x}_k) - f^* \to O(\frac{\alpha}{1-\gamma}).$$

DGD convergence: statement, cont'ed

- In the statement, we observe two things: convergence of the mean values and consensus around the mean, this is a general characteristic of distributed optimization

DGD convergence: statement, cont'ed

- In the statement, we observe two things: convergence of the mean values and consensus around the mean, this is a general characteristic of distributed optimization
- We also see a dependency on the graph entering via its spectrum γ, and in particular (here) the second largest eigenvalue (in modulus).

DGD convergence: statement, cont'ed

- In the statement, we observe two things: convergence of the mean values and consensus around the mean, this is a general characteristic of distributed optimization
- We also see a dependency on the graph entering via its spectrum γ, and in particular (here) the second largest eigenvalue (in modulus).
- The proofs can be extended to cases in which f_i are strongly convex (easier to handle), but it is much harder to work with non-doubly stochastic matrices (i.e., communication issues). See arXiv:1310.7063

DGD convergence: statement, cont'ed

- In the statement, we observe two things: convergence of the mean values and consensus around the mean, this is a general characteristic of distributed optimization
- We also see a dependency on the graph entering via its spectrum γ, and in particular (here) the second largest eigenvalue (in modulus).
- The proofs can be extended to cases in which f_i are strongly convex (easier to handle), but it is much harder to work with non-doubly stochastic matrices (i.e., communication issues). See arXiv:1310.7063
- We can see how for a constant step size we obtain a constant error bound (not so surprising, since we are changing the cost function!)

DGD convergence: statement, cont'ed

- In the statement, we observe two things: convergence of the mean values and consensus around the mean, this is a general characteristic of distributed optimization
- We also see a dependency on the graph entering via its spectrum γ, and in particular (here) the second largest eigenvalue (in modulus).
- The proofs can be extended to cases in which f_i are strongly convex (easier to handle), but it is much harder to work with non-doubly stochastic matrices (i.e., communication issues). See arXiv:1310.7063
- We can see how for a constant step size we obtain a constant error bound (not so surprising, since we are changing the cost function!)
- When α is diminishing, you can obtain a zero error bound, but the analysis is more complicated, and you require typically extra assumptions.

Communication issues

- The parameter γ depends on the graph. For a full graph with weights $1/N$, $\gamma = 0$, because ..

Communication issues

- The parameter γ depends on the graph. For a full graph with weights $1/N$, $\gamma = 0$, because ..
- γ will grow depending on how the graph is connected (in general, the more connected the lower is the γ).

Communication issues

- The parameter γ depends on the graph. For a full graph with weights $1/N$, $\gamma = 0$, because ..
- γ will grow depending on how the graph is connected (in general, the more connected the lower is the γ).
- **Communication issues**

Communication issues

- The parameter γ depends on the graph. For a full graph with weights $1/N$, $\gamma = 0$, because ..
- γ will grow depending on how the graph is connected (in general, the more connected the lower is the γ).
- **Communication issues**
- Directed communication: this is hard, since we loose double-stochasticity. Also you can have cycles in the graph and counter-intuitive behaviors. Usually solved with algorithms of the type "push-sum"

Communication issues

- The parameter γ depends on the graph. For a full graph with weights $1/N$, $\gamma = 0$, because ..
- γ will grow depending on how the graph is connected (in general, the more connected the lower is the γ).
- **Communication issues**
- Directed communication: this is hard, since we loose double-stochasticity. Also you can have cycles in the graph and counter-intuitive behaviors. Usually solved with algorithms of the type "push-sum"
- Package drops, asynchronous communication. This is often less hard to deal with, if the instantaneous W stay symmetric and doubly stochastic. **Typically**, the idea is to define a finite set of communication of matrices $\{W^r\}_r$ and at each time k we randomly pick from them. If each W^r has the same structural properties of W and the eigenvalues of a restricted sequence of W^r have the same properties of the eigenvalues of W, then we can extend convergence results.

Communication issues

- The parameter γ depends on the graph. For a full graph with weights $1/N$, $\gamma = 0$, because ..
- γ will grow depending on how the graph is connected (in general, the more connected the lower is the γ).
- **Communication issues**
- Directed communication: this is hard, since we loose double-stochasticity. Also you can have cycles in the graph and counter-intuitive behaviors. Usually solved with algorithms of the type "push-sum"
- Package drops, asynchronous communication. This is often less hard to deal with, if the instantaneous W stay symmetric and doubly stochastic. **Typically**, the idea is to define a finite set of communication of matrices $\{W^r\}_r$ and at each time k we randomly pick from them. If each W^r has the same structural properties of W and the eigenvalues of a restricted sequence of W^r have the same properties of the eigenvalues of W, then we can extend convergence results.
- Gossip protocols: asynchronous and directed. A node wakes up and transmits to some of its neighbors, it also analyses what it has received, then it sleeps again. This is **very realistic** but **quite hard to do**. Here latencies..

Gradient tracking

- We have seen that DGD has fundamental trade-off between accuracy and performance. Can we do better?

Gradient tracking

- We have seen that DGD has fundamental trade-off between accuracy and performance. Can we do better?
- Consider the following two-communication rounds algorithm

Gradient tracking

- We have seen that DGD has fundamental trade-off between accuracy and performance. Can we do better?
- Consider the following two-communication rounds algorithm
  - **Decentralized Gradient Tracking**

Gradient tracking

- We have seen that DGD has fundamental trade-off between accuracy and performance. Can we do better?
- Consider the following two-communication rounds algorithm

- **Decentralized Gradient Tracking**
  - Initialize $\mathbf{x}_k^i$, $\mathbf{g}_k^i$

Gradient tracking

- We have seen that DGD has fundamental trade-off between accuracy and performance. Can we do better?
- Consider the following two-communication rounds algorithm

- **Decentralized Gradient Tracking**
  - Initialize x_k^i, g_k^i
  - Each device gets and sends x_k^j / x_k^i from/to the neighbors

Gradient tracking

- We have seen that DGD has fundamental trade-off between accuracy and performance. Can we do better?
- Consider the following two-communication rounds algorithm

- **Decentralized Gradient Tracking**
  - Initialize $\mathbf{x}_k^i$, $\mathbf{g}_k^i$
  - Each device gets and sends $\mathbf{x}_k^j / \mathbf{x}_k^i$ from/to the neighbors
  - Each device computes:

$$\mathbf{x}_{k+1}^i = \sum_{j=1}^{N} w_{ij} \mathbf{x}_k^j - \alpha \mathbf{g}_k^i, \quad \forall i$$

Gradient tracking

- We have seen that DGD has fundamental trade-off between accuracy and performance. Can we do better?
- Consider the following two-communication rounds algorithm

- **Decentralized Gradient Tracking**
  - Initialize $\mathbf{x}_k^i$, $\mathbf{g}_k^i$
  - Each device gets and sends $\mathbf{x}_k^j/\mathbf{x}_k^i$ from/to the neighbors
  - Each device computes:

$$\mathbf{x}_{k+1}^i = \sum_{j=1}^{N} w_{ij}\mathbf{x}_k^j - \alpha \mathbf{g}_k^i, \quad \forall i$$

  - Each device gets and sends $\mathbf{g}_k^j/\mathbf{g}_k^i$ from/to the neighbors

Gradient tracking

- We have seen that DGD has fundamental trade-off between accuracy and performance. Can we do better?
- Consider the following two-communication rounds algorithm

Decentralized Gradient Tracking

- Initialize x_k^i, $\boldsymbol{g}_k^i$
- Each device gets and sends x_k^j / x_k^i from/to the neighbors
- Each device computes:

$$x_{k+1}^i = \sum_{j=1}^{N} w_{ij} x_k^j - \alpha \boldsymbol{g}_k^i, \quad \forall i$$

- Each device gets and sends $\boldsymbol{g}_k^j / \boldsymbol{g}_k^i$ from/to the neighbors
- Each device computes:

$$\boldsymbol{g}_{k+1}^i = \sum_{j=1}^{N} w_{ij} \boldsymbol{g}_k^j + (\nabla f_i(x_{k+1}^i) - \nabla f_i(x_k^i)), \quad \forall i$$

Gradient tracking

- We have seen that DGD has fundamental trade-off between accuracy and performance. Can we do better?
- Consider the following two-communication rounds algorithm
  - **Decentralized Gradient Tracking**
    - Initialize $\mathbf{x}_k^i$, $\mathbf{g}_k^i$
    - Each device gets and sends $\mathbf{x}_k^j/\mathbf{x}_k^i$ from/to the neighbors
    - Each device computes:

$$\mathbf{x}_{k+1}^i = \sum_{j=1}^{N} w_{ij}\mathbf{x}_k^j - \alpha \mathbf{g}_k^i, \quad \forall i$$

    - Each device gets and sends $\mathbf{g}_k^j/\mathbf{g}_k^i$ from/to the neighbors
    - Each device computes:

$$\mathbf{g}_{k+1}^i = \sum_{j=1}^{N} w_{ij}\mathbf{g}_k^j + (\nabla f_i(\mathbf{x}_{k+1}^i) - \nabla f_i(\mathbf{x}_k^i)), \quad \forall i$$

- We have now a term that tracks the gradient values and "integrates" the errors

Gradient tracking convergence

- Start by defining the mean quantity, $\bar{x}_k = \frac{1}{N}\sum_{i=1}^{N} x_k^i$

Gradient tracking convergence

- Start by defining the mean quantity, $\bar{x}_k = \frac{1}{N}\sum_{i=1}^{N} x_k^i$

Theorem 5 (Gradient Tracking convergence)

Consider Problem (P) and its solution via a gradient tracking algorithm with constant step size α, and doubly stochastic communication matrix W with $w_{ij} \geq 0$. Let convex functions f_i be L-smooth and strongly convex. If α is chosen small enough and in particular $\leq O(1/L)$, then

1. *Consensus:*

$$\|x_k^i - \bar{x}_k\| \to 0;$$

2. *Convergence*

$$\|\bar{x}_k - x^*\| \to 0.$$

And convergence is linear.

Proof: arXiv:1906.10760

Gradient tracking explanation

- The proof construct a recursion of the form $\boldsymbol{v}_{k+1} \leq J(\alpha)\boldsymbol{v}_k$, and the condition on α guarantees that $\|J(\alpha)\| < 1$.

Gradient tracking explanation

- The proof construct a recursion of the form $\boldsymbol{v}_{k+1} \leq J(\alpha)\boldsymbol{v}_k$, and the condition on α guarantees that $\|J(\alpha)\| < 1$.
- Extensions to non-strongly convex or to other communication patters are possible, but very involved

Gradient tracking explanation

- The proof construct a recursion of the form $\mathbf{v}_{k+1} \leq J(\alpha)\mathbf{v}_k$, and the condition on α guarantees that $\|J(\alpha)\| < 1$.
- Extensions to non-strongly convex or to other communication patters are possible, but very involved
- On the good side, we have a first-order primal method that is fully distributed and allows us to reach zero consensus and residual error.

Gradient tracking explanation

- The proof construct a recursion of the form $\boldsymbol{v}_{k+1} \leq J(\alpha)\boldsymbol{v}_k$, and the condition on α guarantees that $\|J(\alpha)\| < 1$.
- Extensions to non-strongly convex or to other communication patters are possible, but very involved
- On the good side, we have a first-order primal method that is fully distributed and allows us to reach zero consensus and residual error.
- On the minus side, we communicate quite a lot in terms of gradients, so we lose in privacy

What have we learned?

- The general problem of interest is minimizing $\sum_i f_i(x)$

What have we learned?

- The general problem of interest is minimizing $\sum_i f_i(\mathbf{x})$
- When N is big or f_i cannot be shared (efficiently or at all), decentralized schemes play an important role

What have we learned?

- The general problem of interest is minimizing $\sum_i f_i(\mathbf{x})$
- When N is big or f_i cannot be shared (efficiently or at all), decentralized schemes play an important role
- Communication and computation and their trade-off are then key: and of course, first-order algorithms

What have we learned?

- The general problem of interest is minimizing $\sum_i f_i(\mathbf{x})$
- When N is big or f_i cannot be shared (efficiently or at all), decentralized schemes play an important role
- Communication and computation and their trade-off are then key: and of course, first-order algorithms
- Two type of schemes: Incremental or parallel

What have we learned?

- The general problem of interest is minimizing $\sum_i f_i(\mathbf{x})$
- When N is big or f_i cannot be shared (efficiently or at all), decentralized schemes play an important role
- Communication and computation and their trade-off are then key: and of course, first-order algorithms
- Two type of schemes: Incremental or parallel
- For the former, we have seen a convergence result. The latter yields "distributed optimization" and, in the primal space, we have seen two example of algorithms: DGD and Gradient Tracking in a simplified "controlled" setting

What have we learned?

- The general problem of interest is minimizing $\sum_i f_i(\mathbf{x})$
- When N is big or f_i cannot be shared (efficiently or at all), decentralized schemes play an important role
- Communication and computation and their trade-off are then key: and of course, first-order algorithms
- Two type of schemes: Incremental or parallel
- For the former, we have seen a convergence result. The latter yields "distributed optimization" and, in the primal space, we have seen two example of algorithms: DGD and Gradient Tracking in a simplified "controlled" setting
- Convergence proofs in distributed settings are harder since the communication plays an important role!

What have we learned?

- The general problem of interest is minimizing $\sum_i f_i(\mathbf{x})$
- When N is big or f_i cannot be shared (efficiently or at all), decentralized schemes play an important role
- Communication and computation and their trade-off are then key: and of course, first-order algorithms
- Two type of schemes: Incremental or parallel
- For the former, we have seen a convergence result. The latter yields "distributed optimization" and, in the primal space, we have seen two example of algorithms: DGD and Gradient Tracking in a simplified "controlled" setting
- Convergence proofs in distributed settings are harder since the communication plays an important role!
- Why would I care about communication or distributed when we can all upload on the cloud?

What have we learned?

- The general problem of interest is minimizing $\sum_i f_i(\mathbf{x})$
- When N is big or f_i cannot be shared (efficiently or at all), decentralized schemes play an important role
- Communication and computation and their trade-off are then key: and of course, first-order algorithms
- Two type of schemes: Incremental or parallel
- For the former, we have seen a convergence result. The latter yields "distributed optimization" and, in the primal space, we have seen two example of algorithms: DGD and Gradient Tracking in a simplified "controlled" setting
- Convergence proofs in distributed settings are harder since the communication plays an important role!
- Why would I care about communication or distributed when we can all upload on the cloud? (Think BIG: large-data files, server farms, or many sensors, etc..)

Sample references

1. The standard book: *Dimitri P. Bertsekas and John N. Tsitsiklis*, **Parallel and Distributed Computation: Numerical Methods**, 1997, `https://web.mit.edu/dimitrib/www/pdc.html`
2. Two recent articles:
   Kun Yuan, Qing Ling, Wotao Yin, **On the Convergence of Decentralized Gradient Descent**, arXiv:1310.7063 and SIAM Journal on Optimization, 2016
   Giuseppe Notarstefano, Ivano Notarnicola, Andrea Camisa, **Distributed Optimization for Smart Cyber-Physical Networks**, arXiv:1906.10760 and Foundations and Trends in Systems and Control, 2019
3. Many variants out there.

Class 2

Let's revisit our problem

- We remind that the problem at hand is

$$(P) \qquad \min_{x \in \mathrm{R}^n} \sum_{i=1}^{N} f_i(x).$$

and we proceed as in the first class by endowing each device with a copy of x, so that the problem becomes,

$$(P') \qquad \min_{x^i \in \mathrm{R}^n, i=1,\dots,N} \sum_{i=1}^{N} f_i(x^i) \quad \text{subject to } x^i = x^j, \forall i \sim j$$

Let's revisit our problem

- We remind that the problem at hand is

$$(P) \qquad \min_{x \in \mathrm{R}^n} \sum_{i=1}^{N} f_i(x).$$

and we proceed as in the first class by endowing each device with a copy of x, so that the problem becomes,

$$(P') \qquad \min_{x^i \in \mathrm{R}^n, i=1,\dots,N} \sum_{i=1}^{N} f_i(x^i) \quad \text{subject to } x^i = x^j, \forall i \sim j$$

- We can then write (P') compactly as,

$$(P'') \qquad \min_{y \in \mathrm{R}^{Nn}} F(y) \quad \text{subject to } Ay = 0$$

Let's revisit our problem

- We remind that the problem at hand is

$$(P) \qquad \min_{x \in \mathrm{R}^n} \sum_{i=1}^{N} f_i(x).$$

and we proceed as in the first class by endowing each device with a copy of x, so that the problem becomes,

$$(P') \qquad \min_{x^i \in \mathrm{R}^n, i=1,\dots,N} \sum_{i=1}^{N} f_i(x^i) \quad \text{subject to } x^i = x^j, \forall i \sim j$$

- We can then write (P') compactly as,

$$(P'') \qquad \min_{y \in \mathrm{R}^{Nn}} F(y) \quad \text{subject to } Ay = 0$$

- Can we now look at its dual problem?

Let's revisit duality

- Consider the problem

$$\min_{x \in \mathbf{R}^n} f(x) \quad \text{subject to } Ax = b,$$

for a convex function $f : \mathbf{R}^n \to \mathbf{R}$, and matrix $A \in \mathbf{R}^{p \times n}$, vector $b \in \mathbf{R}^p$ (with b in the image of A).

Let's revisit duality

- Consider the problem

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to } Ax = b,$$

for a convex function $f : \mathbf{R}^n \to \mathbf{R}$, and matrix $A \in \mathbf{R}^{p \times n}$, vector $b \in \mathbf{R}^p$ (with b in the image of A).

- The Lagrangian function is defined as,

$$L(x, \lambda) = f(x) + \lambda^\top (Ax - b),$$

where $\lambda \in \mathbf{R}^p$ are the Lagrangian multipliers (aka the dual variables)

Let's revisit duality

- Consider the problem

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to } Ax = b,$$

for a convex function $f : \mathbf{R}^n \to \mathbf{R}$, and matrix $A \in \mathbf{R}^{p \times n}$, vector $b \in \mathbf{R}^p$ (with b in the image of A).

- The Lagrangian function is defined as,

$$L(x, \lambda) = f(x) + \lambda^\top (Ax - b),$$

where $\lambda \in \mathbf{R}^p$ are the Lagrangian multipliers (aka the dual variables)

- The Lagrangian dual function is then,

$$q(\lambda) = \inf_{x \in \mathbb{R}^n} L(x, \lambda) = -(f^\star(-A^\top \lambda) + \lambda^\top b),$$

where $f^\star$ is the conjugate function of f, i.e., $f^\star(y) = \sup_x \{y^\top x - f(x)\}$.
Homework. Prove the last equality.

Dual problems

- The dual problem is then,

$$\max_{\lambda \in \mathbb{R}^p} q(\lambda)$$

Dual problems

- The dual problem is then,
$$\max_{\lambda \in \mathbb{R}^p} q(\lambda)$$
- The dual problem is always convex, even if the primal is not and provide a lower bound on the minimum of the primal problem, i.e.,
$$q^* \leq f^*.$$

Dual problems

- The dual problem is then,

$$\max_{\lambda \in \mathbb{R}^p} q(\lambda)$$

- The dual problem is always convex, even if the primal is not and provide a lower bound on the minimum of the primal problem, i.e.,

$$q^* \leq f^*.$$

- Equality holds if some constraint qualification holds. For convex problems, if Slater's condition is verified (there exists a strictly feasible solution), then equality holds and we say that we have strong duality.

Dual problems

- The dual problem is then,

$$\max_{\lambda \in \mathrm{R}^p} q(\lambda)$$

- The dual problem is always convex, even if the primal is not and provide a lower bound on the minimum of the primal problem, i.e.,

$$q^* \leq f^*.$$

- Equality holds if some constraint qualification holds. For convex problems, if Slater's condition is verified (there exists a strictly feasible solution), then equality holds and we say that we have strong duality.
- For this course strong duality always holds because I don't look at dualizing inequality constraints.

Let's revisit our problem

- We remind that the problem at hand is

$$(P'') \qquad \min_{\boldsymbol{y} \in \mathrm{R}^{Nn}} F(\boldsymbol{y}) \quad \text{subject to } A\boldsymbol{y} = 0$$

whose Lagrangian is

$$L(\boldsymbol{y}, \lambda) = F(\boldsymbol{y}) + \lambda^\top A\boldsymbol{y}$$

and dual problem,

$$\max_{\lambda} q(\lambda) := \inf_{\boldsymbol{y}} \{F(\boldsymbol{y}) + \lambda^\top A\boldsymbol{y}\} = -F^\star(-A^\top \lambda).$$

Let's revisit our problem

- We remind that the problem at hand is

$$(P'') \qquad \min_{\boldsymbol{y}\in\mathbb{R}^{Nn}} F(\boldsymbol{y}) \quad \text{subject to } A\boldsymbol{y} = 0$$

whose Lagrangian is

$$L(\boldsymbol{y}, \lambda) = F(\boldsymbol{y}) + \lambda^\top A\boldsymbol{y}$$

and dual problem,

$$\max_{\lambda} q(\lambda) := \inf_{\boldsymbol{y}} \{F(\boldsymbol{y}) + \lambda^\top A\boldsymbol{y}\} = -F^\star(-A^\top\lambda).$$

- Can we set up a dual ascent algorithm?

Dual ascent = gradient descent for dual

- Start with a λ_0 and iterate:

Dual ascent = gradient descent for dual

- Start with a λ_0 and iterate:
- $\lambda_{k+1} = \lambda_k + \alpha_k \boldsymbol{v}_k$, with $\boldsymbol{v}_k \in \partial q(\lambda_k)$, for all $k = 0, 1, \ldots$.
  Rem: the dual function may not be differentiable even if f is so, so we need the sub-differential

Dual ascent = gradient descent for dual

- Start with a λ_0 and iterate:
- $\lambda_{k+1} = \lambda_k + \alpha_k \boldsymbol{v}_k$, with $\boldsymbol{v}_k \in \partial q(\lambda_k)$, for all $k = 0, 1, \ldots$.
  Rem: the dual function may not be differentiable even if f is so, so we need the sub-differential
- What's $\partial q(\lambda_k)$?

Dual ascent = gradient descent for dual

- Start with a λ_0 and iterate:
- $\lambda_{k+1} = \lambda_k + \alpha_k \boldsymbol{v}_k$, with $\boldsymbol{v}_k \in \partial q(\lambda_k)$, for all $k = 0, 1, \ldots$.
  Rem: the dual function may not be differentiable even if f is so, so we need the sub-differential
- What's $\partial q(\lambda_k)$?
- Start with

$$\partial_\lambda q(\lambda_k) \quad = \quad -A\partial_\lambda[-F^\star(-A^\top \lambda_k)]$$

Then, we know that for convex functions $(\partial_v F)^{-1}(v) = \partial_u F^\star(u)$, and in addition let,

$$\boldsymbol{y}^*(\lambda) := \arg\min_{\boldsymbol{y}} L(\boldsymbol{y}, \lambda) \iff \boldsymbol{y}^*(\lambda) = (\partial F)^{-1}(-A^\top \lambda)$$

Therefore,

$$\partial q(\lambda_k) = A\boldsymbol{y}^*(\lambda_k) \qquad \text{residual map!}$$

Dual decomposition

- Therefore the dual ascent yields, Start with a λ_0 and iterate:

Dual decomposition

- Therefore the dual ascent yields, Start with a λ_0 and iterate:
- Solve locally

$$\mathbf{y}^*(\lambda) := \arg\min_{\mathbf{y}} L(\mathbf{y}, \lambda_k) = \arg\min_{\mathbf{x}^1,\ldots,\mathbf{x}^N} \sum_{i=1}^{n} f_i(\mathbf{x}^i) + \lambda_k^\top A[\mathbf{x}^1, \ldots, \mathbf{x}^N]^\top$$

Upon dividing $A = [A_1|\ldots|A_N]$,

$$\iff \mathbf{x}^{i,*}(\lambda_k) := \arg\min_{\mathbf{x}_i} f_i(\mathbf{x}^i) + \lambda_k^\top A_i \mathbf{x}^i \qquad \forall i,$$

Dual decomposition

- Therefore the dual ascent yields, Start with a λ_0 and iterate:
- Solve locally

$$\boldsymbol{y}^*(\lambda) := \arg\min_{\boldsymbol{y}} L(\boldsymbol{y}, \lambda_k) = \arg\min_{\boldsymbol{x}^1,\ldots,\boldsymbol{x}^N} \sum_{i=1}^{n} f_i(\boldsymbol{x}^i) + \lambda_k^\top A[\boldsymbol{x}^1, \ldots, \boldsymbol{x}^N]^\top$$

Upon dividing $A = [A_1|\ldots|A_N]$,

$$\iff \boldsymbol{x}^{i,*}(\lambda_k) := \arg\min_{\boldsymbol{x}_i} f_i(\boldsymbol{x}^i) + \lambda_k^\top A_i \boldsymbol{x}^i \qquad \forall i,$$

- Send and update $\boldsymbol{x}^{i,*}(\lambda_k)$ to all, and gather,

$$\partial q(\lambda_k) = A\boldsymbol{y}^*(\lambda_k) = \sum_i A_i \boldsymbol{x}^{i,*}(\lambda_k)$$

Dual decomposition

- Therefore the dual ascent yields, Start with a λ_0 and iterate:
- Solve locally

$$\mathbf{y}^*(\lambda) := \arg\min_{\mathbf{y}} L(\mathbf{y}, \lambda_k) = \arg\min_{\mathbf{x}^1,\ldots,\mathbf{x}^N} \sum_{i=1}^{n} f_i(\mathbf{x}^i) + \lambda_k^\top A[\mathbf{x}^1, \ldots, \mathbf{x}^N]^\top$$

Upon dividing $A = [A_1 | \ldots | A_N]$,

$$\iff \mathbf{x}^{i,*}(\lambda_k) := \arg\min_{\mathbf{x}_i} f_i(\mathbf{x}^i) + \lambda_k^\top A_i \mathbf{x}^i \qquad \forall i,$$

- Send and update $\mathbf{x}^{i,*}(\lambda_k)$ to all, and gather,

$$\partial q(\lambda_k) = A\mathbf{y}^*(\lambda_k) = \sum_i A_i \mathbf{x}^{i,*}(\lambda_k)$$

- Update then, $\lambda_{k+1} = \lambda_k + \alpha_k \sum_i A_i \mathbf{x}^{i,*}(\lambda_k)$, for all $k = 0, 1, \ldots$.

Dual decomposition

- The method is called dual decomposition, since you decompose the primal problem via the dual variables;

Dual decomposition

- The method is called dual decomposition, since you decompose the primal problem via the dual variables;
- The dual variables are updated globally, the primal locally, and this is convenient if f_i depends on data that you can't share/ don't want to!

Dual decomposition

- The method is called dual decomposition, since you decompose the primal problem via the dual variables;
- The dual variables are updated globally, the primal locally, and this is convenient if f_i depends on data that you can't share/ don't want to!
- λ can also live locally on the edges!
  Let A be the collection of $x^i - x^j$ for the edge set. Then the associated dual variable is λ^{ij} and that can live on the edge i, j.

Dual decomposition

- The method is called dual decomposition, since you decompose the primal problem via the dual variables;
- The dual variables are updated globally, the primal locally, and this is convenient if f_i depends on data that you can't share/ don't want to!
- λ can also live locally on the edges!
  Let A be the collection of $x^i - x^j$ for the edge set. Then the associated dual variable is λ^{ij} and that can live on the edge i, j.
- The last means that also λ^{ij} can also be done locally!
  $\lambda_{k+1}^{ij} = \lambda_k^{ij} + \alpha_k[x^{i,*}(\lambda_k) - x^{j,*}(\lambda_k)]$, for all $k = 0, 1, \ldots$

Dual decomposition: a picture

- Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, undirected. We indicate $i \sim j$ if there is an edge between i and j. We also define A as

$$A_{ij} = [1_i, -1_j] \qquad \forall i \sim j, \, j < i.$$

Dual decomposition: a picture

- Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, undirected. We indicate $i \sim j$ if there is an edge between i and j. We also define A as

$$A_{ij} = [1_i, -1_j] \qquad \forall i \sim j, \, j < i.$$

- Initialize λ_0^{ij} for $(i, j) \in \mathcal{E}, j < i$

Dual decomposition: a picture

- Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, undirected. We indicate $i \sim j$ if there is an edge between i and j. We also define A as

$$A_{ij} = [1_i, -1_j] \qquad \forall i \sim j,\, j < i.$$

- Initialize λ_0^{ij} for $(i,j) \in \mathcal{E}, j < i$
- For all nodes i do,

$$\mathbf{x}^{i,*}(\lambda_k) := \arg\min_{\mathbf{x}_i} f_i(\mathbf{x}^i) + \sum_{i \sim j} \lambda_k^{ij,\top} A_{ij} \mathbf{x}^i$$

Dual decomposition: a picture

- Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, undirected. We indicate $i \sim j$ if there is an edge between i and j. We also define A as

$$A_{ij} = [1_i, -1_j] \qquad \forall i \sim j, j < i.$$

- Initialize λ_0^{ij} for $(i, j) \in \mathcal{E}, j < i$
- For all nodes i do,

$$\mathbf{x}^{i,*}(\lambda_k) := \arg\min_{\mathbf{x}_i} f_i(\mathbf{x}^i) + \sum_{i \sim j} \lambda_k^{ij,\top} A_{ij} \mathbf{x}^i$$

- Send $\mathbf{x}^{i,*}(\lambda_k)$ to neighbors

Dual decomposition: a picture

- Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, undirected. We indicate $i \sim j$ if there is an edge between i and j. We also define A as

$$A_{ij} = [1_i, -1_j] \qquad \forall i \sim j,\, j < i.$$

- Initialize λ_0^{ij} for $(i, j) \in \mathcal{E}, j < i$
- For all nodes i do,

$$\mathbf{x}^{i,*}(\lambda_k) := \arg\min_{\mathbf{x}_i} f_i(\mathbf{x}^i) + \sum_{i \sim j} \lambda_k^{ij,\top} A_{ij} \mathbf{x}^i$$

- Send $\mathbf{x}^{i,*}(\lambda_k)$ to neighbors
- Update for node i, j:

$$\lambda_{k+1}^{ij} = \lambda_k^{ij} + \alpha_k [\mathbf{x}^{i,*}(\lambda_k) - \mathbf{x}^{j,*}(\lambda_k)]$$

Dual decomposition: a picture

- Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, undirected. We indicate $i \sim j$ if there is an edge between i and j. We also define A as

$$A_{ij} = [1_i, -1_j] \qquad \forall i \sim j,\, j < i.$$

- Initialize λ_0^{ij} for $(i, j) \in \mathcal{E}, j < i$
- For all nodes i do,

$$\mathbf{x}^{i,*}(\lambda_k) := \arg\min_{\mathbf{x}_i} f_i(\mathbf{x}^i) + \sum_{i \sim j} \lambda_k^{ij,\top} A_{ij} \mathbf{x}^i$$

- Send $\mathbf{x}^{i,*}(\lambda_k)$ to neighbors
- Update for node i, j:

$$\lambda_{k+1}^{ij} = \lambda_k^{ij} + \alpha_k [\mathbf{x}^{i,*}(\lambda_k) - \mathbf{x}^{j,*}(\lambda_k)]$$

- This has two local updates and one communication round (synchronous, undirected)

Dual decomposition: convergence I

- We look now at the convergence of dual decomposition. We need to recall some of the properties of the dual function. In particular, let $\sigma(A)$ be the singular values of A.
  If f is m-strongly convex then $-q$ is $\sigma^2_{\max}(A)/m$ smooth;
  If f is L-smooth then $-q$ is $\sigma^2_{\min}(A)/L$ strongly convex;
  Homework. Remind yourself of why it is so.

Dual decomposition: convergence I

- We look now at the convergence of dual decomposition. We need to recall some of the properties of the dual function. In particular, let $\sigma(A)$ be the singular values of A.
  If f is m-strongly convex then $-q$ is $\sigma^2_{\max}(A)/m$ smooth;
  If f is L-smooth then $-q$ is $\sigma^2_{\min}(A)/L$ strongly convex;
  Homework. Remind yourself of why it is so.
- Then we have the following

Dual decomposition: convergence I

- We look now at the convergence of dual decomposition. We need to recall some of the properties of the dual function. In particular, let $\sigma(A)$ be the singular values of A.
  If f is m-strongly convex then $-q$ is $\sigma^2_{\max}(A)/m$ smooth;
  If f is L-smooth then $-q$ is $\sigma^2_{\min}(A)/L$ strongly convex;
  Homework. Remind yourself of why it is so.
- Then we have the following

Theorem 6 (Dual decomposition convergence)

Consider problem (P) and the dual decomposition approach for a certain connection matrix A. If f_i's are m-strongly convex and L-smooth, then we can select $\alpha < 2m/\sigma^2_{\max}(A)$ and obtaining a linear rate of convergence as,

$$\|\lambda_k - \lambda^*\| \leq \left[\max\{|1 - \alpha\frac{\sigma^2_{\max}(A)}{m}|, |1 - \alpha\frac{\sigma^2_{\min}(A)}{L}|\}\right]^k \|\lambda_0 - \lambda^*\|$$

and

$$\|x^i_k - x^*\| \leq \frac{\sigma_{\max}(A)}{m}\|\lambda_k - \lambda^*\|.$$

Dual decomposition: convergence II

- Proof follows from OPT201/202. **Homework.** Work it out.

Dual decomposition: convergence II

- Proof follows from OPT201/202. **Homework.** Work it out.
- Convergence is proven when A is full row rank (so that $\sigma_{\min} > 0$). But why would it be?

Dual decomposition: convergence II

- Proof follows from OPT201/202. **Homework.** Work it out.
- Convergence is proven when A is full row rank (so that $\sigma_{\min} > 0$). But why would it be?
- In the case A is not full row rank? λ^* is not unique and $-q$ is not strongly convex. But, we can look at restricted strong convexity instead !

Dual decomposition: convergence II

- Proof follows from OPT201/202. **Homework.** Work it out.
- Convergence is proven when A is full row rank (so that $\sigma_{\min} > 0$). But why would it be?
- In the case A is not full row rank? λ^* is not unique and $-q$ is not strongly convex. But, we can look at restricted strong convexity instead !

Let's see how to do it

Dual decomposition: convergence II

- Proof follows from OPT201/202. **Homework.** Work it out.
- Convergence is proven when A is full row rank (so that $\sigma_{\min} > 0$). But why would it be?
- In the case A is not full row rank? λ^* is not unique and $-q$ is not strongly convex. But, we can look at restricted strong convexity instead !

Let's see how to do it

- First $\lambda^* = \lambda_0^* + \lambda_1^*$, with $\lambda_0^* \in \mathrm{im}(A)$ and $\lambda_1^* \in \mathrm{null}(A^\top)$. One can show that λ_0^* is unique and we concentrate on that one, since λ_1^* is redundant $A^\top \lambda_0^* = 0$.

Dual decomposition: convergence II

- Proof follows from OPT201/202. **Homework.** Work it out.
- Convergence is proven when A is full row rank (so that $\sigma_{\min} > 0$). But why would it be?
- In the case A is not full row rank? λ^* is not unique and $-q$ is not strongly convex. But, we can look at restricted strong convexity instead !

Let's see how to do it

- First $\lambda^* = \lambda_0^* + \lambda_1^*$, with $\lambda_0^* \in \mathrm{im}(A)$ and $\lambda_1^* \in \mathrm{null}(A^\top)$. One can show that λ_0^* is unique and we concentrate on that one, since λ_1^* is redundant $A^\top \lambda_0^* = 0$.
- Start with $\lambda_0 \in \mathrm{im}(A)$; then, it is direct that:

$$\lambda_{k+1} = \lambda_k + \alpha A \mathbf{x} \in \mathrm{im}(A)$$

Dual decomposition: convergence II

- Proof follows from OPT201/202. **Homework.** Work it out.
- Convergence is proven when A is full row rank (so that $\sigma_{\min} > 0$). But why would it be?
- In the case A is not full row rank? λ^* is not unique and $-q$ is not strongly convex. But, we can look at restricted strong convexity instead !

Let's see how to do it

- First $\lambda^* = \lambda_0^* + \lambda_1^*$, with $\lambda_0^* \in \mathrm{im}(A)$ and $\lambda_1^* \in \mathrm{null}(A^\top)$. One can show that λ_0^* is unique and we concentrate on that one, since λ_1^* is redundant $A^\top \lambda_0^* = 0$.
- Start with $\lambda_0 \in \mathrm{im}(A)$; then, it is direct that:

$$\lambda_{k+1} = \lambda_k + \alpha A \mathbf{x} \in \mathrm{im}(A)$$

- Then, can we say that (restricted strong convexity)

$$(\partial q(\lambda) - \partial q(\lambda'))^\top(\lambda' - \lambda) \geq \sigma_{\underline{\min}}^2/L\|\lambda' - \lambda\|^2, \quad \forall \lambda, \lambda' \in \mathrm{im}(A)?$$

with $\sigma_{\underline{\min}}$ the minimum non-zero singular value of A?

Dual decomposition: convergence II

- Yes, substitute $\partial q(\lambda) = A \partial F^\star(-A^\top \lambda)$, then,

$$\begin{aligned}(\partial F^\star(-A^\top\lambda) - \partial F^\star(-A^\top\lambda'))^\top A^\top(\lambda' - \lambda) = \\ (\partial F^\star(-A^\top\lambda) - \partial F^\star(-A^\top\lambda'))^\top(-A^\top\lambda + A^\top\lambda') \geq \\ 1/L\|A^\top(\lambda' - \lambda)\|^2 \geq \sigma^2_{\min}/L\|\lambda' - \lambda)\|^2.\end{aligned}$$

Dual decomposition: convergence II

- Yes, substitute $\partial q(\lambda) = A\partial F^\star(-A^\top\lambda)$, then,

$$\begin{aligned}(\partial F^\star(-A^\top\lambda) - \partial F^\star(-A^\top\lambda'))^\top A^\top(\lambda' - \lambda) = \\ (\partial F^\star(-A^\top\lambda) - \partial F^\star(-A^\top\lambda'))^\top(-A^\top\lambda + A^\top\lambda') \geq \\ 1/L\|A^\top(\lambda' - \lambda)\|^2 \geq \sigma^2_{\min}/L\|\lambda' - \lambda)\|^2.\end{aligned}$$

- The latest step is due to the fact that $A^\top(\lambda - \lambda') = 0$ iff $\lambda = \lambda'$.

Dual decomposition: convergence II

- Yes, substitute $\partial q(\lambda) = A\partial F^\star(-A^\top\lambda)$, then,

$$
\begin{aligned}
(\partial F^\star(-A^\top\lambda) - \partial F^\star(-A^\top\lambda'))^\top A^\top(\lambda' - \lambda) = \\
(\partial F^\star(-A^\top\lambda) - \partial F^\star(-A^\top\lambda'))^\top(-A^\top\lambda + A^\top\lambda') \geq \\
1/L\|A^\top(\lambda' - \lambda)\|^2 \geq \sigma^2_{\underline{\mathrm{min}}}/L\|\lambda' - \lambda)\|^2.
\end{aligned}
$$

- The latest step is due to the fact that $A^\top(\lambda - \lambda') = 0$ iff $\lambda = \lambda'$.
- So the Theorem becomes: we can select $\alpha < 2m/\sigma^2_{\mathrm{max}}(A)$ and $\lambda_0 \in \mathrm{im}(A)$, and obtaining a linear rate of convergence as,

$$
\|\lambda_k - \lambda^*\| \leq \left[\max\{|1 - \alpha\frac{\sigma^2_{\mathrm{max}}(A)}{m}|, |1 - \alpha\frac{\sigma^2_{\underline{\mathrm{min}}}(A)}{L}|\}\right]^k \|\lambda_0 - \lambda^*\|
$$

Dual decomposition: convergence II

- Yes, substitute $\partial q(\lambda) = A\partial F^\star(-A^\top\lambda)$, then,

$$
\begin{aligned}
&(\partial F^\star(-A^\top\lambda) - \partial F^\star(-A^\top\lambda'))^\top A^\top(\lambda' - \lambda) = \\
&\qquad (\partial F^\star(-A^\top\lambda) - \partial F^\star(-A^\top\lambda'))^\top(-A^\top\lambda + A^\top\lambda') \geq \\
&\qquad\qquad 1/L\|A^\top(\lambda' - \lambda)\|^2 \geq \sigma^2_{\underline{\min}}/L\|\lambda' - \lambda)\|^2.
\end{aligned}
$$

- The latest step is due to the fact that $A^\top(\lambda - \lambda') = 0$ iff $\lambda = \lambda'$.
- So the Theorem becomes: we can select $\alpha < 2m/\sigma^2_{\max}(A)$ and $\lambda_0 \in \mathrm{im}(A)$, and obtaining a linear rate of convergence as,

$$
\|\lambda_k - \lambda^*\| \leq \left[\max\{|1 - \alpha\frac{\sigma^2_{\max}(A)}{m}|, |1 - \alpha\frac{\sigma^2_{\underline{\min}}(A)}{L}|\}\right]^k \|\lambda_0 - \lambda^*\|
$$

- Note: $\lambda_0 = \mathbf{0} \in \mathrm{im}(A)$

Dual decomposition: convergence III

- The step size can be chosen constant and you still converge linearly ! (As in the gradient tracking case); but the step size has to be chosen carefully

Dual decomposition: convergence III

- The step size can be chosen constant and you still converge linearly ! (As in the gradient tracking case); but the step size has to be chosen carefully
- Convergence depends on the graph, and in particular on $\sigma_{\min}, \sigma_{\max}$
  It is not only a matter of function properties, but also graph properties!

Dual decomposition: convergence III

- The step size can be chosen constant and you still converge linearly ! (As in the gradient tracking case); but the step size has to be chosen carefully
- Convergence depends on the graph, and in particular on $\sigma_{\min}, \sigma_{\max}$
  It is not only a matter of function properties, but also graph properties!
- Convergence is more tricky when F is not strongly convex, since q will not be differentiable, so it also hard to recover the primal solution here

Dual decomposition: convergence III

- The step size can be chosen constant and you still converge linearly ! (As in the gradient tracking case); but the step size has to be chosen carefully
- Convergence depends on the graph, and in particular on $\sigma_{\min}, \sigma_{\max}$
  It is not only a matter of function properties, but also graph properties!
- Convergence is more tricky when F is not strongly convex, since q will not be differentiable, so it also hard to recover the primal solution here
- Convergence may be complicated when communication is directed, or asynchronous, or you have latencies, as in the first class

The Alternating Direction Method of Multipliers (ADMM)

- To overcome the drawbacks of dual decomposition, we introduce another iterative method: the ADMM. We are going to develop it for a general case and then focus it on our distributed problem

The Alternating Direction Method of Multipliers (ADMM)

- To overcome the drawbacks of dual decomposition, we introduce another iterative method: the ADMM. We are going to develop it for a general case and then focus it on our distributed problem
- Consider the problem:

$$\begin{array}{ll} \min\limits_{x \in \mathbb{R}^n, y \in \mathbb{R}^p} & f(x) + g(y) \\ \text{subject to} & Ax + By = c. \end{array}$$

For well-defined matrices and vectors, as well as convex functions f, g.

The Alternating Direction Method of Multipliers (ADMM)

- To overcome the drawbacks of dual decomposition, we introduce another iterative method: the ADMM. We are going to develop it for a general case and then focus it on our distributed problem
- Consider the problem:

$$\begin{aligned} \min_{\mathbf{x}\in\mathrm{R}^n, \mathbf{y}\in\mathrm{R}^p} \quad & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} \quad & A\mathbf{x} + B\mathbf{y} = c. \end{aligned}$$

  For well-defined matrices and vectors, as well as convex functions f, g.
- Define the augmented Lagrangian,

$$L(\mathbf{x}, \mathbf{y}, \lambda) := f(\mathbf{x}) + g(\mathbf{y}) + \lambda^\top(A\mathbf{x} + B\mathbf{y} - c) + \frac{\beta}{2}\|A\mathbf{x} + B\mathbf{y} - c\|^2,$$

  for any scalar $\beta > 0$.

ADMM: iterations

- Then the ADMM does: start with an initial value for $\boldsymbol{x}_0, \boldsymbol{y}_0, \lambda_0$,

ADMM: iterations

- Then the ADMM does: start with an initial value for $\boldsymbol{x}_0, \boldsymbol{y}_0, \lambda_0$,
- Iterate:

$$\begin{aligned}
\boldsymbol{x}_{k+1} &= \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} L(\boldsymbol{x}, \boldsymbol{y}_k, \lambda_k) \\
\boldsymbol{y}_{k+1} &= \arg\min_{\boldsymbol{y}\in\mathbb{R}^p} L(\boldsymbol{x}_{k+1}, \boldsymbol{y}, \lambda_k) \\
\lambda_{k+1} &= \lambda_{k+1} + \beta(A\boldsymbol{x}_{k+1} + B\boldsymbol{y}_{k+1} - c)
\end{aligned}$$

ADMM: iterations

- Then the ADMM does: start with an initial value for $\boldsymbol{x}_0, \boldsymbol{y}_0, \lambda_0$,
- Iterate:

$$
\begin{aligned}
\boldsymbol{x}_{k+1} &= \arg\min_{\boldsymbol{x}\in\mathrm{R}^n} L(\boldsymbol{x}, \boldsymbol{y}_k, \lambda_k) \\
\boldsymbol{y}_{k+1} &= \arg\min_{\boldsymbol{y}\in\mathrm{R}^p} L(\boldsymbol{x}_{k+1}, \boldsymbol{y}, \lambda_k) \\
\lambda_{k+1} &= \lambda_{k+1} + \beta(A\boldsymbol{x}_{k+1} + B\boldsymbol{y}_{k+1} - c)
\end{aligned}
$$

- ADMM is similar to an incremental approach, but on two variables. It is also similar to dual decomposition, but on two variables

ADMM: iterations

- Then the ADMM does: start with an initial value for $\mathbf{x}_0, \mathbf{y}_0, \lambda_0$,
- Iterate:

$$\begin{aligned}\mathbf{x}_{k+1} &= \arg\min_{\mathbf{x}\in\mathbb{R}^n} L(\mathbf{x}, \mathbf{y}_k, \lambda_k)\\ \mathbf{y}_{k+1} &= \arg\min_{\mathbf{y}\in\mathbb{R}^p} L(\mathbf{x}_{k+1}, \mathbf{y}, \lambda_k)\\ \lambda_{k+1} &= \lambda_{k+1} + \beta(A\mathbf{x}_{k+1} + B\mathbf{y}_{k+1} - c)\end{aligned}$$

- ADMM is similar to an incremental approach, but on two variables. It is also similar to dual decomposition, but on two variables
- It looks more complicated than dual ascent, is it better in some sense?

ADMM: iterations

- Then the ADMM does: start with an initial value for $\mathbf{x}_0, \mathbf{y}_0, \lambda_0$,
- Iterate:

$$
\begin{aligned}
\mathbf{x}_{k+1} &= \arg\min_{\mathbf{x}\in\mathbb{R}^n} L(\mathbf{x}, \mathbf{y}_k, \lambda_k) \\
\mathbf{y}_{k+1} &= \arg\min_{\mathbf{y}\in\mathbb{R}^p} L(\mathbf{x}_{k+1}, \mathbf{y}, \lambda_k) \\
\lambda_{k+1} &= \lambda_{k+1} + \beta(A\mathbf{x}_{k+1} + B\mathbf{y}_{k+1} - c)
\end{aligned}
$$

- ADMM is similar to an incremental approach, but on two variables. It is also similar to dual decomposition, but on two variables
- It looks more complicated than dual ascent, is it better in some sense?
- Assume strong duality holds and primal and dual solutions exist, then convergence is ensure for any $\beta > 0$ in a weak sense.

ADMM: convergence in some special cases

- Assume f to be m-strongly convex and L-smooth and that a primal-dual solution exists (g can be a generic convex function). This is a special case and ADMM generally converges with minimal assumptions, but in a much weaker sense, so we keep here the stronger assumptions.

ADMM: convergence in some special cases

- Assume f to be m-strongly convex and L-smooth and that a primal-dual solution exists (g can be a generic convex function). This is a special case and ADMM generally converges with minimal assumptions, but in a much weaker sense, so we keep here the stronger assumptions.
- Assume that A is full row rank.

ADMM: convergence in some special cases

- Assume f to be m-strongly convex and L-smooth and that a primal-dual solution exists (g can be a generic convex function). This is a special case and ADMM generally converges with minimal assumptions, but in a much weaker sense, so we keep here the stronger assumptions.
- Assume that A is full row rank.
- Then ADMM converges linearly for any step size $\beta > 0$.

ADMM: convergence in some special cases

- Assume f to be m-strongly convex and L-smooth and that a primal-dual solution exists (g can be a generic convex function). This is a special case and ADMM generally converges with minimal assumptions, but in a much weaker sense, so we keep here the stronger assumptions.
- Assume that A is full row rank.
- Then ADMM converges linearly for any step size $\beta > 0$.

Theorem 7 (ADMM convergence)

With the assumptions in place,

$$\|\lambda_k - \lambda^*\| \leq \varrho^k \|\lambda_0 - \lambda^*\| \qquad \varrho = \max\left\{\left|\frac{1-\beta\frac{\sigma^2_{\max}(A)}{m}}{1+\beta\frac{\sigma^2_{\max}(A)}{m}}\right|, \left|\frac{1-\beta\frac{\sigma^2_{\min}(A)}{L}}{1+\beta\frac{\sigma^2_{\min}(A)}{L}}\right|\right\}$$

and,

$$\|x_k - x^*\| \leq \frac{\|A\|}{m}\|\lambda_k - \lambda^*\|.$$

Proof?

- ADMM is extremely popular, for its loose assumptions and the fact that it works in many contexts, even when we have no idea why (non-convex, combinatorial, etc..)

Proof?

- ADMM is extremely popular, for its loose assumptions and the fact that it works in many contexts, even when we have no idea why (non-convex, combinatorial, etc..)
- Hence, we have many ways to prove its convergence. Here, I want to give a glimpse of an operator-splitting technique.

Proof?

- ADMM is extremely popular, for its loose assumptions and the fact that it works in many contexts, even when we have no idea why (non-convex, combinatorial, etc..)
- Hence, we have many ways to prove its convergence. Here, I want to give a glimpse of an operator-splitting technique.
- The proof goes as follows: ADMM is an application of a special algorithm (the Douglas-Rachford splitting) applied to the dual of our initial problem. The Douglas-Rachford splitting converges in a certain way given functional properties. We derive the dual of those functional properties and (as in the dual decomposition case) the convergence of the dual algorithm (ADMM).

Proof?

- ADMM is extremely popular, for its loose assumptions and the fact that it works in many contexts, even when we have no idea why (non-convex, combinatorial, etc..)
- Hence, we have many ways to prove its convergence. Here, I want to give a glimpse of an operator-splitting technique.
- The proof goes as follows: ADMM is an application of a special algorithm (the Douglas-Rachford splitting) applied to the dual of our initial problem. The Douglas-Rachford splitting converges in a certain way given functional properties. We derive the dual of those functional properties and (as in the dual decomposition case) the convergence of the dual algorithm (ADMM).
- This is why the result looks very similar to the result of the dual decomposition. ADMM is a dual algorithm.

Proof: step I, Douglas-Rachford splitting

- Consider the problem,

$$\min_{x \in \mathrm{R}^n} f(x) + g(x),$$

with f and g convex closed and proper (CCP). Consider now the following method to find a solution x^*.

Proof: step I, Douglas-Rachford splitting

- Consider the problem,

$$\min_{x \in \mathrm{R}^n} f(x) + g(x),$$

with f and g convex closed and proper (CCP). Consider now the following method to find a solution x^*.

- Start at a certain z_0 and iterate for all $k \in \mathbb{N}$:

$$x_k = \mathrm{prox}_{\beta f}(z_k) \tag{6a}$$

$$z_{k+1} = z_k + \mathrm{prox}_{\beta g}(2x_k - z_k) - x_k, \tag{6b}$$

where $\mathrm{prox}_{\beta\phi}$ is the usual prox operator:

$$\mathrm{prox}_{\beta\phi}(u) = \arg\min_v \{\phi(v) + \frac{1}{2\beta}\|v - u\|^2\}$$

The method is called the Douglas-Rachford splitting and it converges in some defined sense.

Proof: step I, Douglas-Rachford splitting

- Consider the problem,

$$\min_{x \in \mathbb{R}^n} f(x) + g(x),$$

with f and g convex closed and proper (CCP). Consider now the following method to find a solution x^*.

- Start at a certain z_0 and iterate for all $k \in \mathbb{N}$:

$$x_k = \mathrm{prox}_{\beta f}(z_k) \tag{6a}$$

$$z_{k+1} = z_k + \mathrm{prox}_{\beta g}(2x_k - z_k) - x_k, \tag{6b}$$

where $\mathrm{prox}_{\beta\phi}$ is the usual prox operator:

$$\mathrm{prox}_{\beta\phi}(u) = \arg\min_{v}\{\phi(v) + \frac{1}{2\beta}\|v-u\|^2\}$$

The method is called the Douglas-Rachford splitting and it converges in some defined sense.

- In particular if f is m-strongly convex and L-smooth, then, for all $\beta > 0$

$$\|z_{k+1} - z^*\| \leq \varrho^k \|z_k - z^*\|, \qquad \varrho = \max\left\{\left|\frac{1-\beta L}{1+\beta L}\right|, \left|\frac{1-\beta m}{1+\beta m}\right|\right\}$$

Proof: step II, The dual problem and its properties

- Start from our problem:

$$\begin{array}{ll} \min_{\mathbf{x}\in\mathbb{R}^n,\mathbf{y}\in\mathbb{R}^p} & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} & A\mathbf{x} + B\mathbf{y} = c, \end{array}$$

  form the Lagrangian and take the dual

Proof: step II, The dual problem and its properties

- Start from our problem:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n,\mathbf{y}\in\mathbb{R}^p} \quad & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} \quad & A\mathbf{x} + B\mathbf{y} = c, \end{aligned}$$

form the Lagrangian and take the dual

- Dual problem is,

$$\max_{\lambda} \inf_{\mathbf{x},\mathbf{y}} \{f(\mathbf{x}) + g(\mathbf{y}) + \lambda^\top(A\mathbf{x} + B\mathbf{y} - c)\}$$

and so,

$$\min_{\lambda} f^\star(-A^\top\lambda) + g^\star(-A^\top\lambda) + c^\top\lambda$$

Proof: step II, The dual problem and its properties

- Start from our problem:

$$\begin{aligned} \min_{x\in\mathbb{R}^n, y\in\mathbb{R}^p} \quad & f(x)+g(y) \\ \text{subject to} \quad & Ax+By=c, \end{aligned}$$

  form the Lagrangian and take the dual

- Dual problem is,

$$\max_{\lambda} \inf_{x,y} \{f(x)+g(y)+\lambda^\top(Ax+By-c)\}$$

  and so,

$$\min_{\lambda} f^\star(-A^\top\lambda)+g^\star(-A^\top\lambda)+c^\top\lambda$$

- Consider "f" $= f^\star(-A^\top\lambda)+c^\top\lambda$ and "g" $= g^\star(-B^\top\lambda)$

Proof: step II, The dual problem and its properties

- Start from our problem:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n, \mathbf{y}\in\mathbb{R}^p} \quad & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} \quad & A\mathbf{x} + B\mathbf{y} = c, \end{aligned}$$

form the Lagrangian and take the dual

- Dual problem is,

$$\max_{\lambda} \inf_{\mathbf{x},\mathbf{y}} \{ f(\mathbf{x}) + g(\mathbf{y}) + \lambda^\top (A\mathbf{x} + B\mathbf{y} - c) \}$$

and so,

$$\min_{\lambda} f^\star(-A^\top\lambda) + g^\star(-A^\top\lambda) + c^\top\lambda$$

- Consider "f"$= f^\star(-A^\top\lambda) + c^\top\lambda$ and "g"$= g^\star(-B^\top\lambda)$
- As before we know that "f" is $\sigma^2_{\min}/L$-strongly convex and $\sigma^2_{\max}/m$-smooth

Proof: step III, applying DR to the dual

- Apply DRS "f" $= f^\star(-A^\top\lambda) + c^\top\lambda$ and "g" $= g^\star(-B^\top\lambda)$ and use its convergence properties.

Proof: step III, applying DR to the dual

- Apply DRS "f" $= f^\star(-A^\top \lambda) + c^\top \lambda$ and "g" $= g^\star(-B^\top \lambda)$ and use its convergence properties.
- **Do some computational gymnastic: check out the references**

Proof: step III, applying DR to the dual

- Apply DRS "f"$= f^\star(-A^\top \lambda) + c^\top \lambda$ and "g"$= g^\star(-B^\top \lambda)$ and use its convergence properties.
- **Do some computational gymnastic: check out the references**
- Arrive at the ADMM iterations

ADMM: distributed optimization

- Let's go back to:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n, \mathbf{y}\in\mathbb{R}^p} \quad & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} \quad & A\mathbf{x} + B\mathbf{y} = c, \end{aligned}$$

and our problem,

$$\begin{aligned} \min_{\mathbf{x}^i\in\mathbb{R}^n, \mathbf{y}^{ij}\in\mathbb{R}^{|\mathcal{E}|}} \quad & \sum_{i=1}^{N} f_i(\mathbf{x}^i) \\ \text{subject to} \quad & \mathbf{x}^i = \mathbf{y}^{ij}, \qquad \forall\, i \sim j. \end{aligned}$$

ADMM: distributed optimization

- Let's go back to:

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathrm{R}^n, \boldsymbol{y}\in\mathrm{R}^p} \quad & f(\boldsymbol{x}) + g(\boldsymbol{y}) \\ \text{subject to} \quad & A\boldsymbol{x} + B\boldsymbol{y} = c, \end{aligned}$$

  and our problem,

$$\begin{aligned} \min_{\boldsymbol{x}^i\in\mathrm{R}^n, \boldsymbol{y}^{ij}\in\mathrm{R}^{|\mathcal{E}|}} \quad & \sum_{i=1}^{N} f_i(\boldsymbol{x}^i) \\ \text{subject to} \quad & \boldsymbol{x}^i = \boldsymbol{y}^{ij}, \qquad \forall\, i \sim j. \end{aligned}$$

- Think a bit at what is written. This is of course not the only possibility. Different splittings yield different properties. This will give a distributed algorithm similar to the one we have seen for dual decomposition.

ADMM: distributed optimization

- Let's go back to:

$$\begin{aligned} \min_{\mathbf{x}\in\mathrm{R}^n,\mathbf{y}\in\mathrm{R}^p} \quad & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} \quad & A\mathbf{x} + B\mathbf{y} = c, \end{aligned}$$

  and our problem,

$$\begin{aligned} \min_{\mathbf{x}^i\in\mathrm{R}^n,\mathbf{y}^{ij}\in\mathrm{R}^{|\mathcal{E}|}} \quad & \sum_{i=1}^N f_i(\mathbf{x}^i) \\ \text{subject to} \quad & \mathbf{x}^i = \mathbf{y}^{ij}, \qquad \forall\, i \sim j. \end{aligned}$$

- Think a bit at what is written. This is of course not the only possibility. Different splittings yield different properties. This will give a distributed algorithm similar to the one we have seen for dual decomposition.
- Later, we will see a different cloud-based splitting instead.

ADMM: distributed optimization

- Let's go back to:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n, \mathbf{y}\in\mathbb{R}^p} \quad & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} \quad & A\mathbf{x} + B\mathbf{y} = c, \end{aligned}$$

  and our problem,

$$\begin{aligned} \min_{\mathbf{x}^i\in\mathbb{R}^n, \mathbf{y}^{ij}\in\mathbb{R}^{|\mathcal{E}|}} \quad & \sum_{i=1}^{N} f_i(\mathbf{x}^i) \\ \text{subject to} \quad & \mathbf{x}^i = \mathbf{y}^{ij}, \qquad \forall\, i \sim j. \end{aligned}$$

- Think a bit at what is written. This is of course not the only possibility. Different splittings yield different properties. This will give a distributed algorithm similar to the one we have seen for dual decomposition.
- Later, we will see a different cloud-based splitting instead.
- Compactify $\mathbf{x}^i = \mathbf{y}^{ij}$ as, $A\mathbf{x} + B\mathbf{y} = 0$. Careful here that A is not full row rank, so linear convergence requires more work, but possible.

ADMM: distributed algorithm

- Then you apply ADMM and simplify,

ADMM: distributed algorithm

- Then you apply ADMM and simplify,
- For all nodes update:

$$x^i_{k+1} = \arg\min_{x^i}\{f_i(x) + (\lambda_k)^\top(Ax + By_k) + \frac{\beta}{2}\|Ax + By_k\|^2\}$$

which is equivalent to,

$$x^i_{k+1} = \arg\min_{x^i}\{f_i(x^i) + \sum_{j\in\mathcal{N}^i}\frac{\beta}{2}\|x^i - y^{ij}_k + \frac{\lambda^{ij}_k}{\beta}\|^2\}$$

ADMM: distributed algorithm

- Then you apply ADMM and simplify,
- For all nodes update:

$$x^i_{k+1} = \arg\min_{x^i}\{f_i(x) + (\lambda_k)^\top(Ax + By_k) + \frac{\beta}{2}\|Ax + By_k\|^2\}$$

which is equivalent to,

$$x^i_{k+1} = \arg\min_{x^i}\{f_i(x^i) + \sum_{j\in\mathcal{N}^i}\frac{\beta}{2}\|x^i - y^{ij}_k + \frac{\lambda^{ij}_k}{\beta}\|^2\}$$

- Communicate x^i_{k+1} to your neighbors

ADMM: distributed algorithm

- Then you apply ADMM and simplify,
- For all nodes update:

$$\mathbf{x}^i_{k+1} = \arg\min_{\mathbf{x}^i}\{f_i(\mathbf{x}) + (\lambda_k)^\top(A\mathbf{x} + B\mathbf{y}_k) + \frac{\beta}{2}\|A\mathbf{x} + B\mathbf{y}_k\|^2\}$$

which is equivalent to,

$$\mathbf{x}^i_{k+1} = \arg\min_{\mathbf{x}^i}\{f_i(\mathbf{x}^i) + \sum_{j\in\mathcal{N}^i}\frac{\beta}{2}\|\mathbf{x}^i - \mathbf{y}^{ij}_k + \frac{\lambda^{ij}_k}{\beta}\|^2\}$$

- Communicate $\mathbf{x}^i_{k+1}$ to your neighbors
- Each node,

$$\mathbf{y}^{ij}_{k+1} = \arg\min_{\mathbf{y}^{ij}}\{(\lambda_k)^\top(A\mathbf{x}_{k+1} + B\mathbf{y}) + \frac{\beta}{2}\|A\mathbf{x}_{k+1} + B\mathbf{y}\|^2\}$$

which is equivalent to,

$$\mathbf{y}^{ij}_{k+1} = \arg\min_{\mathbf{y}^{ij}}\{\frac{\beta}{2}\|\mathbf{x}^i_{k+1} - \mathbf{y}^{ij} + \frac{\lambda^{ij}_k}{\beta}\|^2 + \frac{\beta}{2}\|\mathbf{x}^j_{k+1} - \mathbf{y}^{ij} + \frac{\lambda^{ij}_k}{\beta}\|^2\}$$

ADMM: distributed algorithm cont'ed

- $$\mathbf{y}^{ij}_{k+1} = \arg\min_{\mathbf{y}^{ij}} \{ \frac{\beta}{2} \| \mathbf{x}^i_{k+1} - \mathbf{y}^{ij} + \frac{\lambda^{ij}_k}{\beta} \|^2 + \frac{\beta}{2} \| \mathbf{x}^j_{k+1} - \mathbf{y}^{ij} + \frac{\lambda^{ij}_k}{\beta} \|^2 \}$$

  which is

  $$\mathbf{y}^{ij}_{k+1} = \frac{\mathbf{x}^i_{k+1} + \mathbf{x}^j_{k+1}}{2} + \frac{\lambda^{ij}_k}{\beta}$$

ADMM: distributed algorithm cont'ed

- $$y^{ij}_{k+1} = \arg\min_{y^{ij}} \{ \frac{\beta}{2} \| x^i_{k+1} - y^{ij} + \frac{\lambda^{ij}_k}{\beta} \|^2 + \frac{\beta}{2} \| x^j_{k+1} - y^{ij} + \frac{\lambda^{ij}_k}{\beta} \|^2 \}$$

  which is

  $$y^{ij}_{k+1} = \frac{x^i_{k+1} + x^j_{k+1}}{2} + \frac{\lambda^{ij}_k}{\beta}$$

- Each node,

  $$\lambda^{ij}_{k+1} = \lambda^{ij}_k + \beta(x^i_{k+1} + x^j_{k+1} - 2y^{ij}_{k+1}) = 0$$

ADMM: distributed algorithm cont'ed

- $$\boldsymbol{y}^{ij}_{k+1} = \arg\min_{\boldsymbol{y}^{ij}}\{\frac{\beta}{2}\|\boldsymbol{x}^i_{k+1} - \boldsymbol{y}^{ij} + \frac{\lambda^{ij}_k}{\beta}\|^2 + \frac{\beta}{2}\|\boldsymbol{x}^j_{k+1} - \boldsymbol{y}^{ij} + \frac{\lambda^{ij}_k}{\beta}\|^2\}$$

  which is

  $$\boldsymbol{y}^{ij}_{k+1} = \frac{\boldsymbol{x}^i_{k+1} + \boldsymbol{x}^j_{k+1}}{2} + \frac{\lambda^{ij}_k}{\beta}$$

- Each node,

  $$\lambda^{ij}_{k+1} = \lambda^{ij}_k + \beta(\boldsymbol{x}^i_{k+1} + \boldsymbol{x}^j_{k+1} - 2\boldsymbol{y}^{ij}_{k+1}) = 0$$

1. Start with $\lambda^{ij}_0 = 0$, and $\boldsymbol{y}^{ij}_0$ then,
2. Each node updates, $\boldsymbol{x}^i_{k+1} = \arg\min_{\boldsymbol{x}^i}\{f_i(\boldsymbol{x}^i) + \sum_{j\in\mathcal{N}^i}\frac{\beta}{2}\|\boldsymbol{x}^i - \boldsymbol{y}^{ij}_k\|^2\}$
3. Each node communicates $\boldsymbol{x}^i_{k+1}$ to its neighbors
4. Each node updates, $\boldsymbol{y}^{ij}_{k+1} = \frac{\boldsymbol{x}^i_{k+1}+\boldsymbol{x}^j_{k+1}}{2}$

ADMM: Example in model fitting

- Consider the task of training a model via the convex problem,

$$\min_{x \in \mathbb{R}^n} \ell(Ax - b) + r(x),$$

where $\ell : \mathbf{R}^p \to \mathbf{R}$ is a convex loss function, $A \in \mathbf{R}^{p \times n}$ is the feature matrix, $b \in \mathbf{R}^p$ is the output vector, x are the parameters of the model, and $r : \mathbf{R}^n \to \mathbf{R}$ is a convex regularization function.

ADMM: Example in model fitting

- Consider the task of training a model via the convex problem,

$$\min_{x \in \mathrm{R}^n} \ell(Ax - b) + r(x),$$

where $\ell : \mathbf{R}^p \to \mathbf{R}$ is a convex loss function, $A \in \mathbf{R}^{p \times n}$ is the feature matrix, $b \in \mathbf{R}^p$ is the output vector, x are the parameters of the model, and $r : \mathbf{R}^n \to \mathbf{R}$ is a convex regularization function.

- Assume that ℓ is additive across training examples, as

$$\ell(x) = \sum_{i=1}^{m} \ell_i(a_i^\top x - b_i).$$

ADMM: Example in model fitting

- Consider the task of training a model via the convex problem,

$$\min_{x \in \mathrm{R}^n} \ell(Ax - b) + r(x),$$

where $\ell : \mathbf{R}^p \to \mathbf{R}$ is a convex loss function, $A \in \mathbf{R}^{p \times n}$ is the feature matrix, $b \in \mathbf{R}^p$ is the output vector, x are the parameters of the model, and $r : \mathbf{R}^n \to \mathbf{R}$ is a convex regularization function.

- Assume that ℓ is additive across training examples, as

$$\ell(x) = \sum_{i=1}^{m} \ell_i(a_i^\top x - b_i).$$

- r is often separable too. For instance if r is the Tikhonov regularization (aka ridge penalty): $r(x) = \nu \|x\|_2^2$, or if r is the lasso penalty: $r(x) = \nu \|x\|_1$

ADMM: Example in model fitting

- Consider the task of training a model via the convex problem,

$$\min_{x \in \mathbb{R}^n} \ell(Ax - b) + r(x),$$

where $\ell : \mathbf{R}^p \to \mathbf{R}$ is a convex loss function, $A \in \mathbf{R}^{p \times n}$ is the feature matrix, $b \in \mathbf{R}^p$ is the output vector, x are the parameters of the model, and $r : \mathbf{R}^n \to \mathbf{R}$ is a convex regularization function.

- Assume that ℓ is additive across training examples, as

$$\ell(x) = \sum_{i=1}^{m} \ell_i(a_i^\top x - b_i).$$

- r is often separable too. For instance if r is the Tikhonov regularization (aka ridge penalty): $r(x) = \nu\|x\|_2^2$, or if r is the lasso penalty: $r(x) = \nu\|x\|_1$
- Typically you have a modest number of features but a very large number of training examples (i.e., $m \gg n$).

ADMM: Example in model fitting II

- The goal is to solve the problem in a distributed way, with each processor handling a subset of the training data. This is useful either when there are so many training examples that it is inconvenient or impossible to process them on a single machine or when the data is naturally collected or stored in a distributed fashion. This includes, for example, online social network data, webserver access logs, wireless sensor networks, and many cloud computing applications more generally.

ADMM: Example in model fitting II

- The goal is to solve the problem in a distributed way, with each processor handling a subset of the training data. This is useful either when there are so many training examples that it is inconvenient or impossible to process them on a single machine or when the data is naturally collected or stored in a distributed fashion. This includes, for example, online social network data, webserver access logs, wireless sensor networks, and many cloud computing applications more generally.
- Suppose you associate a number of training examples to a number N of processors (in the extreme $N = m$). Then, you may solve

$$\begin{aligned} \min_{\mathbf{x}^j \in \mathbf{R}^n, \mathbf{y} \in \mathbf{R}^n} \quad & \sum_{j=1}^{N} \ell_j(A_j \mathbf{x}^j - b_j) + r(\mathbf{y}), \\ \text{subject to} \quad & \mathbf{x}^j = \mathbf{y}, \quad i = 1, \ldots, N. \end{aligned}$$

ADMM: Example in model fitting II

- The goal is to solve the problem in a distributed way, with each processor handling a subset of the training data. This is useful either when there are so many training examples that it is inconvenient or impossible to process them on a single machine or when the data is naturally collected or stored in a distributed fashion. This includes, for example, online social network data, webserver access logs, wireless sensor networks, and many cloud computing applications more generally.
- Suppose you associate a number of training examples to a number N of processors (in the extreme $N = m$). Then, you may solve

$$\begin{aligned} \min_{\boldsymbol{x}^j \in \mathrm{R}^n, \boldsymbol{y} \in \mathrm{R}^n} \quad & \sum_{j=1}^{N} \ell_j(A_j \boldsymbol{x}^j - b_j) + r(\boldsymbol{y}), \\ \text{subject to} \quad & \boldsymbol{x}^j = \boldsymbol{y}, \quad i = 1, \ldots, N. \end{aligned}$$

- This can be solved via ADMM

ADMM: Example in model fitting III

- Each processor solves

$$x_{k+1}^j = \arg\min_{x^j} \{\ell_j(A_j x^j - b^j) + \lambda_k^j (x^j - y_k) + \frac{\beta}{2}\|x^j - y_k\|^2\}$$

$$= \arg\min_{x^j} \{\ell_j(A_j x^j - b^j) + \frac{\beta}{2}\|x^j - y_k + \frac{\lambda_k^j}{\beta}\|^2\}$$

ADMM: Example in model fitting III

- Each processor solves

$$\mathbf{x}^j_{k+1} = \arg\min_{\mathbf{x}^j} \{\ell_j(A_j\mathbf{x}^j - b^j) + \lambda^j_k(\mathbf{x}^j - \mathbf{y}_k) + \frac{\beta}{2}\|\mathbf{x}^j - \mathbf{y}_k\|^2\}$$

$$= \arg\min_{\mathbf{x}^j} \{\ell_j(A_j\mathbf{x}^j - b^j) + \frac{\beta}{2}\|\mathbf{x}^j - \mathbf{y}_k + \frac{\lambda^j_k}{\beta}\|^2\}$$

- Communication to the cloud of $\mathbf{x}^j_{k+1}$

ADMM: Example in model fitting III

- Each processor solves

$$
\begin{aligned}
x^j_{k+1} &= \arg\min_{x^j}\{\ell_j(A_j x^j - b^j) + \lambda^j_k(x^j - y_k) + \frac{\beta}{2}\|x^j - y_k\|^2\} \\
&= \arg\min_{x^j}\{\ell_j(A_j x^j - b^j) + \frac{\beta}{2}\|x^j - y_k + \frac{\lambda^j_k}{\beta}\|^2\}
\end{aligned}
$$

- Communication to the cloud of x^j_{k+1}
- On the cloud we solve,

$$
\begin{aligned}
y_{k+1} &= \arg\min_{y}\{r(y) + \sum_{j=1}^{N}\lambda^j_k(x^j_{k+1} - y) + \frac{\beta}{2}\|x^j_{k+1} - y\|^2\} \\
&= \arg\min_{y}\{r(y) + \sum_{j=1}^{N}\frac{\beta}{2}\|x^j_{k+1} - y - \frac{\lambda^j_k}{\beta}\|^2\}
\end{aligned}
$$

ADMM: Example in model fitting III

- Each processor solves

$$\mathbf{x}^j_{k+1} = \arg\min_{\mathbf{x}^j}\{\ell_j(A_j\mathbf{x}^j - b^j) + \lambda^j_k(\mathbf{x}^j - \mathbf{y}_k) + \frac{\beta}{2}\|\mathbf{x}^j - \mathbf{y}_k\|^2\}$$
$$= \arg\min_{\mathbf{x}^j}\{\ell_j(A_j\mathbf{x}^j - b^j) + \frac{\beta}{2}\|\mathbf{x}^j - \mathbf{y}_k + \frac{\lambda^j_k}{\beta}\|^2\}$$

- Communication to the cloud of $\mathbf{x}^j_{k+1}$
- On the cloud we solve,

$$\mathbf{y}_{k+1} = \arg\min_{\mathbf{y}}\{r(\mathbf{y}) + \sum_{j=1}^N \lambda^j_k(\mathbf{x}^j_{k+1} - \mathbf{y}) + \frac{\beta}{2}\|\mathbf{x}^j_{k+1} - \mathbf{y}\|^2\}$$
$$= \arg\min_{\mathbf{y}}\{r(\mathbf{y}) + \sum_{j=1}^N \frac{\beta}{2}\|\mathbf{x}^j_{k+1} - \mathbf{y} - \frac{\lambda^j_k}{\beta}\|^2\}$$

- Update $\lambda^j_{k+1} = \lambda^j_k + \beta(\mathbf{x}^j_{k+1} - \mathbf{y}_{k+1})$ and communicate back to processors $\mathbf{y}_{k+1}, \lambda^j_{k+1}$

ADMM: Example in model fitting IV

- The data never leaves the processors

ADMM: Example in model fitting IV

- The data never leaves the processors
- Different updates of $\boldsymbol{y}$ depending on r and on $\boldsymbol{x}$ depending on ℓ.

ADMM: Example in model fitting IV

- The data never leaves the processors
- Different updates of $\boldsymbol{y}$ depending on r and on $\boldsymbol{x}$ depending on ℓ.
- Consider for example: $\ell(\cdot) = \|\cdot\|_2^2$, $r(\boldsymbol{y}) = \nu\|\boldsymbol{y}\|_1$, then we obtain the algorithm,

ADMM: Example in model fitting IV

- The data never leaves the processors
- Different updates of $\boldsymbol{y}$ depending on r and on $\boldsymbol{x}$ depending on ℓ.
- Consider for example: $\ell(\cdot) = \|\cdot\|_2^2$, $r(\boldsymbol{y}) = \nu\|\boldsymbol{y}\|_1$, then we obtain the algorithm,
- Each processor solves

$$\boldsymbol{x}_{k+1}^j = \arg\min_{\boldsymbol{x}^j}\{\|A_j\boldsymbol{x}^j - b_j\|^2) + \frac{\beta}{2}\|\boldsymbol{x}^j - \boldsymbol{y}_k + \frac{\lambda_k^j}{\beta}\|^2\}$$
$$= (A_j^\top A_j + \beta I)^{-1}(A_j^\top b_j - \beta\boldsymbol{y}_k + \lambda_k^j).$$

ADMM: Example in model fitting IV

- The data never leaves the processors
- Different updates of $\boldsymbol{y}$ depending on r and on $\boldsymbol{x}$ depending on ℓ.
- Consider for example: $\ell(\cdot) = \|\cdot\|_2^2$, $r(\boldsymbol{y}) = \nu\|\boldsymbol{y}\|_1$, then we obtain the algorithm,
- Each processor solves

$$\boldsymbol{x}^j_{k+1} = \arg\min_{\boldsymbol{x}^j}\{\|A_j\boldsymbol{x}^j - b_j\|^2) + \frac{\beta}{2}\|\boldsymbol{x}^j - \boldsymbol{y}_k + \frac{\lambda^j_k}{\beta}\|^2\}$$
$$= (A_j^\top A_j + \beta I)^{-1}(A_j^\top b_j - \beta\boldsymbol{y}_k + \lambda^j_k).$$

- Communication to the cloud of $\boldsymbol{x}^j_{k+1}$

ADMM: Example in model fitting IV

- The data never leaves the processors
- Different updates of $\boldsymbol{y}$ depending on r and on $\boldsymbol{x}$ depending on ℓ.
- Consider for example: $\ell(\cdot) = \|\cdot\|_2^2$, $r(\boldsymbol{y}) = \nu\|\boldsymbol{y}\|_1$, then we obtain the algorithm,
- Each processor solves

$$\begin{aligned} \boldsymbol{x}_{k+1}^j &= \arg\min_{\boldsymbol{x}^j}\{\|A_j\boldsymbol{x}^j - b_j\|^2) + \frac{\beta}{2}\|\boldsymbol{x}^j - \boldsymbol{y}_k + \frac{\lambda_k^j}{\beta}\|^2\} \\ &= (A_j^\top A_j + \beta I)^{-1}(A_j^\top b_j - \beta\boldsymbol{y}_k + \lambda_k^j). \end{aligned}$$

- Communication to the cloud of $\boldsymbol{x}_{k+1}^j$
- On the cloud we solve,

$$\boldsymbol{y}_{k+1} = \arg\min_{\boldsymbol{y}}\{\nu\|\boldsymbol{y}\|_1 + \sum_{j=1}^N \frac{\beta}{2}\|\boldsymbol{x}_{k+1}^j - \boldsymbol{y} + \frac{\lambda_k^j}{\beta}\|^2\} = S_{\nu/\beta}(\bar{\boldsymbol{x}}_{k+1} + \frac{\bar{\lambda}_k}{\beta})$$

ADMM: Example in model fitting IV

- The data never leaves the processors
- Different updates of $\boldsymbol{y}$ depending on r and on $\boldsymbol{x}$ depending on ℓ.
- Consider for example: $\ell(\cdot) = \|\cdot\|_2^2$, $r(\boldsymbol{y}) = \nu\|\boldsymbol{y}\|_1$, then we obtain the algorithm,
- Each processor solves

$$\boldsymbol{x}^j_{k+1} = \arg\min_{\boldsymbol{x}^j}\{\|A_j\boldsymbol{x}^j - b_j\|^2) + \frac{\beta}{2}\|\boldsymbol{x}^j - \boldsymbol{y}_k + \frac{\lambda^j_k}{\beta}\|^2\}$$
$$= (A_j^\top A_j + \beta I)^{-1}(A_j^\top b_j - \beta\boldsymbol{y}_k + \lambda^j_k).$$

- Communication to the cloud of $\boldsymbol{x}^j_{k+1}$
- On the cloud we solve,

$$\boldsymbol{y}_{k+1} = \arg\min_{\boldsymbol{y}}\{\nu\|\boldsymbol{y}\|_1 + \sum_{j=1}^N \frac{\beta}{2}\|\boldsymbol{x}^j_{k+1} - \boldsymbol{y} + \frac{\lambda^j_k}{\beta}\|^2\} = S_{\nu/\beta}(\bar{\boldsymbol{x}}_{k+1} + \frac{\bar{\lambda}_k}{\beta})$$

- Update $\lambda^j_{k+1} = \lambda^j_k + \beta(\boldsymbol{x}^j_{k+1} - \boldsymbol{y}_{k+1})$ and communicate back to processors $\boldsymbol{y}_{k+1}, \lambda^j_{k+1}$

Communication issues

- In the description above, we have used bi-direction synchronous communication, this may not be the case in reality

Communication issues

- In the description above, we have used bi-direction synchronous communication, this may not be the case in reality
- Asynchronicity, latencies, package losses, all add to the difficulty in proving convergence of the algorithm

Communication issues

- In the description above, we have used bi-direction synchronous communication, this may not be the case in reality
- Asynchronicity, latencies, package losses, all add to the difficulty in proving convergence of the algorithm
- Research in this domain is very rich and active!

What have we learned?

- We have looked at distributed optimization in the dual domain, with two "classical" algorithms: dual decomposition and the ADMM

What have we learned?

- We have looked at distributed optimization in the dual domain, with two "classical" algorithms: dual decomposition and the ADMM
- We have studied their convergence, which seem better (in some sense) than primal methods

What have we learned?

- We have looked at distributed optimization in the dual domain, with two "classical" algorithms: dual decomposition and the ADMM
- We have studied their convergence, which seem better (in some sense) than primal methods
- ADMM in particular offers multiple ways to distributed computations across multiple devices and it has received a lot of attention in recent years

What have we learned?

- We have looked at distributed optimization in the dual domain, with two "classical" algorithms: dual decomposition and the ADMM
- We have studied their convergence, which seem better (in some sense) than primal methods
- ADMM in particular offers multiple ways to distributed computations across multiple devices and it has received a lot of attention in recent years
- ADMM works by splitting the problem into a part that can be solved locally, and a part that we can afford to solve sharing information

What have we learned?

- We have looked at distributed optimization in the dual domain, with two "classical" algorithms: dual decomposition and the ADMM
- We have studied their convergence, which seem better (in some sense) than primal methods
- ADMM in particular offers multiple ways to distributed computations across multiple devices and it has received a lot of attention in recent years
- ADMM works by splitting the problem into a part that can be solved locally, and a part that we can afford to solve sharing information
- ADMM can be applied to many settings (multi-core, cloud-computing, distributed computing, etc..)

Sample references

1. *Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein*, **Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers**, Foundations and Trend in Machine Learning, 2010
2. *Ernest Ryu, Stephen Boyd*, **A Primer on Monotone Operator Methods**, Appl. Comput. Math., 2016
3. Many variants out there.