Alternating Direction Method of Multipliers for Convex Optimization in Machine Learning - Interpretation and Implementation

Kuan-Min Huang
Department of Electrical Engineering
National Taipei University
New Taipei City, Taiwan
guanmin60@gmail.com

Hooman Samani
School of Engineering, Computing and Mathematics
University of Plymouth
Plymouth, Devon PL4 8AA, UK
hooman.samani@plymouth.ac.uk

Chan-Yun Yang
Department of Electrical Engineering
National Taipei University
New Taipei City, Taiwan
cyyang@mail.ntpu.edu.tw

Jie-Sheng Chen
Department of Electrical Engineering
National Taipei University
New Taipei City, Taiwan
jerry.870413@yahoo.com.tw

2022 2nd International Conference on Image Processing and Robotics (ICIPRob) | 978-1-6654-0771-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICIPRob54042.2022.9798720

***Abstract*—The alternating direction method of multipliers (ADMM) is an important method to solve convex optimization problems. Due to the optimization tasks increased with the sort of machine learning applications, ADMM has gained much more attention recently. The principle of ADMM solves problems by breaking them into smaller pieces to specially limit the problem dimension. Each of the pieces are then easier to handle and speed up accordingly the total computational time to reach the optimum. With the speeding-up, it was widely adopted for optimization in a number of areas. In this paper, we start the explanation from the constrained convex optimization, and the relation between primal problem and dual problem. With the preliminary explanation, two optimization algorithms are introduced, including the dual ascent and the dual decomposition approaches. An introduction of augmented Lagrangian, the key to success ADMM, is also followed up ahead for elaboration. Finally, the main topic of ADMM is explained algorithmically based on the fundamentals, and an example code is outlined for implementation.**

Keywords—dual problem, dual ascent, convex optimization

I. Introductions

Alternating Direction Method of Multipliers (ADMM), as a statistical learning method, is a computational framework for solving optimization problems, suitable for solving decentralized convex optimization problems. ADMM uses the decomposition-coordination process to decompose a globally large problem into multiple domestic small slices of sub-problems which are relatively easier to solve. The solution of the global problem can then be obtained by coordinating the derived solutions of the sub-problems. ADMM was first proposed by Glowinski and Marrocco [1] and Gabay and Mercier [2] in 1975 and 1976, respectively. One important review published by Boyd *et al.* in 2011 proved that it is suitable for large-scale decentralized optimization problems [2]. Because it was proposed earlier than the submission for the large-scale distributed computing systems, ADMM was never widely known before 2011.

The distributed optimization algorithm is actually the core of distributed machine learning algorithms. It integrates many classic optimization ideas, and combines the ideas for solving the problems encountered in the area of recent learning topics. A more general and efficient solution was proposed and then realized with better implementations. The ADMM algorithm structure in its nature is suitable for solving large-scale machine learning problems which are critical in the following issues: the capability when learning with big data, the capability when learning a scaling up model and the productivity requested by a more practical environment. There is no doubt that large-scale machine learning requires greater data throughput, and processing and learning capabilities than a stand-alone machine learning. For sure we are about to have the ability to learn more complex models, and get better performance of a task with the distributed computing which is deployed with enhanced storage and computing resources.

Without exception, most of the large-scale machine learning task learns its task by the idea of "divide and conquer" *i.e.*, splitting the task into multiple sub-tasks and getting backward the individual solutions generated from the sub-tasks. It means how to solve the problem in the split-merge strategy is conceptually important. ADMM is fantastic in itself the methodological intuition.

For optimization, an convex optimization problems of the form $\min(f(x) + g(z))$ subject to $Ax + Bz = c$ where A and B are coefficient matrices and c is a vector, for the optimum solution (x_{opt}, z_{opt}). It is especially useful when the distributed convex minimization problem rises up to manipulate with an arbitrarily large scale data set. As the problem addressed, we are about to find $x_{\text{opt}} \in X$ such that:

$$f(x_{\text{opt}}) = min\{f(x): x \in X\}, \tag{1}$$

given a constraint $Ax = b$, where $X \in \mathbb{R}^n$is called the feasible set, $f(x): \mathbb{R}^n \mapsto \mathbb{R}$ is the objective function, X and f are convex, matrix $A \in \mathbb{R}^{m\times n}$ and vector $b \in \mathbb{R}^m b \in \mathbb{R}^m$. Here, the input x here may have a huge amount of variables, and the associated coefficient matrix A for the input x can enormously be hundreds of millions of entries long. In such extreme cases, the traditional techniques for minimization may be too slow, despite how fast they may be on normal sized problems. Such issues are solved by using parallel versions of algorithms to dispatch the workload across multiple processors, thus speeding up the optimization. Unfortunately, the traditional optimization algorithms are not suitable for parallel computing, so an alternative variant capable to dispatch the workload becomes sensible. Such a variant would be able to decentralize the optimization. It's why ADMM is getting popular.

We will first give some backgrounds on ADMM, then describe how it works, how it is used to solve problems in practice, and how it can be implemented by an illustrative Matlab example. The result would show the convex optimization algorithm ADMM, splitting the problem into smaller pieces that can be optimized in parallel, is indeed robust.

II. CONSTRAINED CONVEX OPTIMIZATION

A. Definition of Convex Set

Definition: If for any $x, y \in C, \theta \in \mathbb{R}$ with $\theta x + (1-\theta)y \in C$ where $0 \le \theta \le 1$, then a set C is **convex**.

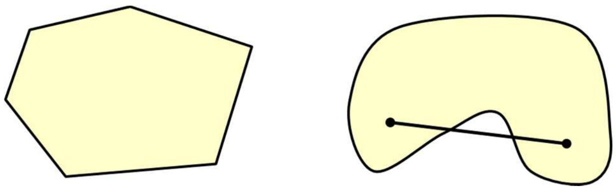


Figure 1. Convex set versus non-convex set.

B. Definition of Convex Function

Definition: A function $f: \mathbb{R}^n \to \mathbb{R}$ is **convex** if its domain is a **convex set**, *i.e.,* $(\theta x + (1-\theta)y) \le \theta f(x) + (1-\theta) f(y)$, with $0 \le \theta \le 1, \theta \in \mathbb{R}$.

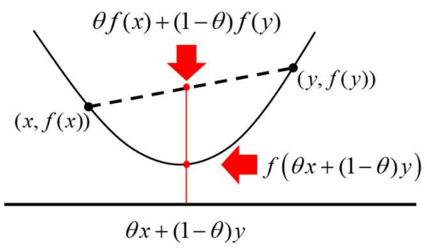


Figure 2. Property of a convex function.

C. Convex Optimization Problem

A convex optimization problem is a problem where all of the constraints are convex functions, and its objective is a convex function when minimization, otherwise a concave function when maximization. Here, an objective minimization is illustrated as follows:

$$\min f(x)$$

$$\text{subject to } \; Ax = b$$

where $x \in \mathbb{R}^n$, , $A \in \mathbb{R}^{m \times n}$, and $f: \mathbb{R}^n \to \mathbb{R}$ is a convex function. A convex optimization problem is usually solved by the method of Lagrange multiplier where the Lagrange form is denoted by

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

where λ is a Lagrange multiplier.

III. PRIMAL PROBLEM AND DUAL PROBLEM

A. Definition of Conjugate Function

Definition: *If* $f: \mathbb{R}^n \to \mathbb{R}$, then

$$f^*(y) = \sup_x (y^T x - f(x))$$

where $f^*(y)$ is the **conjugate function** of $f(x)$.

Remark: f^* is convex even if f is not convex.

Here, we take $f(x) = x^2$ as an example for examination. If $f(x) = x^2$, then

$$f^*(y) = \sup_x \left(y^T x - f(x)\right)$$

$$= \sup_x (xy - x^2)$$

The supremum is attained where the derivative of $xy - x^2$ with respect to x is 0

$$\frac{\partial (xy - x^2)}{\partial x} = 0$$

and get $y = 2x$. Hence we have

$$f^*(y) = \frac{y}{2} \cdot y - \left(\frac{y}{2}\right)^2 = \frac{y^2}{4}$$

B. Definition of Lagrange Dual Form

If the **Lagrange form** of $f(x)$ is

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

then the Lagrange dual form is

$$g(\lambda) = \inf_x L(x, \lambda) = -f^*(-A^T \lambda) - \lambda^T b$$

Proof:

$$\begin{aligned} g(\lambda) &= inf_x \, L(x, \lambda) \\ &= inf_x (f(x) + \lambda^T (Ax - b)) \\ &= inf_x (f(x) + \lambda^T Ax - \lambda^T b)) \\ &= -\lambda^T b + inf_x (f(x) + \lambda^T Ax) \\ &= -\lambda^T b + inf_x - \left(-(f(x) + \lambda^T Ax)\right) \\ &= -\lambda^T b - sup_x (-\lambda^T Ax - f(x)) \\ &= -\lambda^T b - sup_x ((-A^T \lambda)^T x - f(x)) \\ &= -\lambda^T b - f^*(-A^T \lambda) \end{aligned}$$

C. Dual Problem

Consider the following equality-constrained convex optimization problem:

$$\min_x f(x)$$

$$\text{subject to } Ax = b$$

This is referred to as the primal problem (for a primal function *f*), and *x* is referred to as the primal variable. To help in solving this problem, we change the formulation to a different form using the Lagrangian and solve that. The Lagrange form is defined as:

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

We call the dual Lagrange form $g(\lambda) = \inf_x L(x,\lambda)$ and the dual problem $\max_\lambda(g(\lambda))$ where λ is the dual variable. With this formulation, we can get:

$$x_{\text{opt}} = arg \min_x L(x, \lambda_{\text{opt}})$$

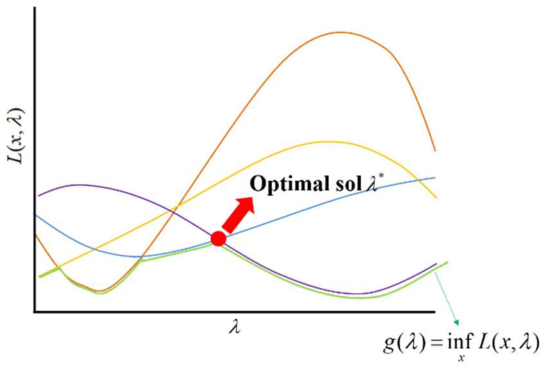
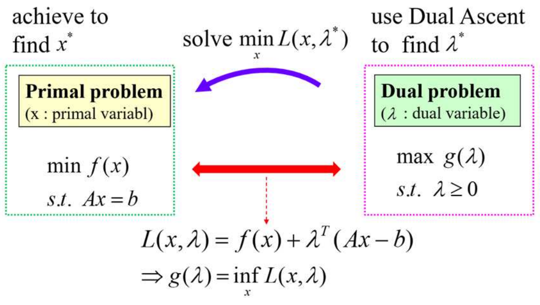


Figure 3. Graph of Lagrange dual form.

and use figure below to simply explain solving process relation between primal problem and dual problem.



achieve to find x^*
solve $\min_x L(x,\lambda^*)$
use Dual Ascent to find λ^*
Primal problem
(x : primal variabl)
$\min f(x)$
$s.t.\ Ax = b$
Dual problem
(λ : dual variable)
$\max g(\lambda)$
$s.t.\ \lambda \geq 0$
$L(x,\lambda) = f(x) + \lambda^T(Ax-b)$
$\Rightarrow g(\lambda) = \inf_x L(x,\lambda)$

Figure 4. Solving process relation between primal problem and dual problem.

IV. Dual Ascent and Dual Decomposition

A. *Dual Ascent Method* (DAM)

One method that gives us solution of dual problem is the Dual Ascent Method (DAM). We use gradient method for dual problem:

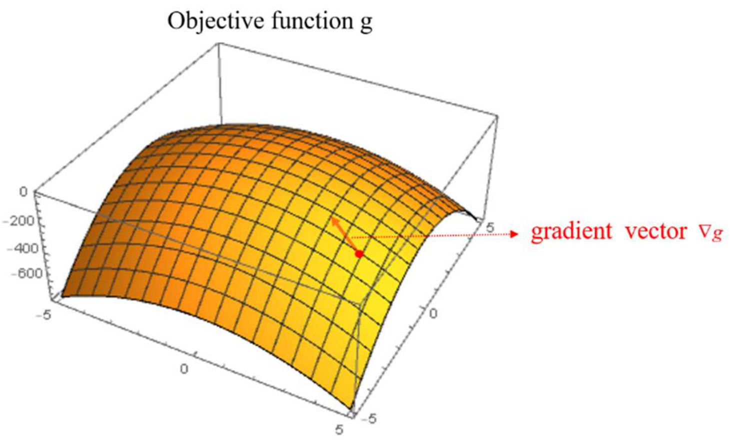
$$\lambda_n = \lambda_{n-1} + \alpha_{n-1}\nabla_g(\lambda_{n-1})$$

We start with iteration number $n = 1$ and start point λ_0, following below steps and computing:

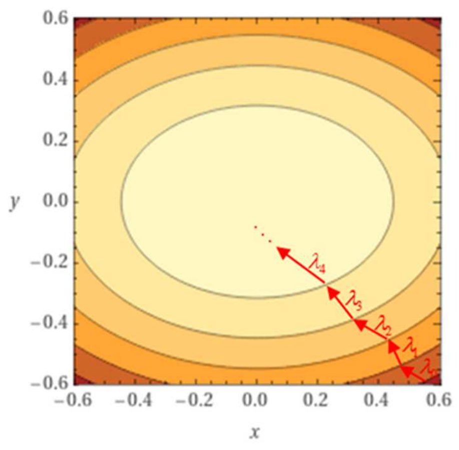
1. Searching direction: $\nabla g(\lambda_{n-1})$ (gradient vector).
2. The step length: Find α_{n-1} such that

$$g(\lambda_{n-1} + \alpha_{n-1}\nabla g(\lambda_{n-1})) > g(\lambda_{n-1})$$

3. Updating λ: set $\lambda_n = \lambda_{n-1} + \alpha_{n-1}\nabla g(\lambda_{n-1})$, and go iteratively back to step 1 until *k* iterations.
4. After *k* iterations, the optimal argument is achieved by setting $\lambda_k = \lambda_{\text{opt}}$



(a) The surface of objective function *g*



(b) The *k*-iteration steps descending the contour curves of its objective function to reach λ_{opt}

Figure 5. We plot 3d objective function and its contour as a Dual Ascent Method example.

Assume both x_k and λ_k are the ***k*-iteration solutions** of primal problem and dual problem, respectively. If λ_k is optimal of dual problem, then optimal solution of primal problem is

$$x_{k+1} = arg \min_x L(x, \lambda_k)$$

DAM is characterized at iteration *k* by computing until convergence:

$$\begin{cases} \lambda_k = \lambda_{k-1} + \alpha_{k-1}\nabla g(\lambda_{k-1}) \\ x_{k+1} = arg \min_x L(x, \lambda_k) \end{cases}, k = 1,2, \ldots$$

It conducts the DAM algorithm shown as Algorithm 1 below:

Algorithm 1. Dual Ascent Method (DAM)

```
1: Initialize  dual variable $\lambda_0$ 、$x_0$
2: repeat
3:     for each iteration do
4:         $x_{k+1} \leftarrow arg\,\min_x L(x_k, \lambda_k)$
5:         $\nabla g(\lambda) = Ax_{k+1} - b$
6:         $\lambda_{k+1} \leftarrow \lambda_k + \alpha_k \nabla g(\lambda)$
7:     end for
8: until get $\lambda_{opt}$
```

B. Dual Decomposition (DD)

Suppose that our objective is **separable**, *i.e.*

$$f(x) = \sum_{i=1}^{N} f_i(x_i),\ x = (x_1, x_2, \cdots, x_n)^T$$

Then we can take the same **separable** idea for the Lagrange form, *i.e.*

$$\begin{aligned} L(x,\lambda) &= \sum_{i=1}^{N} L_i(x_i,\lambda) \\ &= \sum_{i=1}^{N} \left(f_i(x_i) + \lambda^T \left(A_i x_i - \frac{1}{N} b\right) \right) \end{aligned}$$

Thus, our *x*-minimization step in the DAM is split into *n* separate minimizations that can be carried out in parallel, and characterized at iteration *k* by computing until convergence:

$$\begin{cases} \lambda^{(k)} = \lambda^{(k-1)} + \alpha^{(k-1)} \nabla g(\lambda^{(k-1)}) \\ x_i{}^{(k+1)} = arg\,\min_{x_i} L_i\,(x_i, \lambda^{(k)}) \end{cases}$$

for $i = 1,2,\cdots,N$, and $k = 1,2,\cdots$. The algorithm computes the above *x*-minimization step for $\forall i$, in parallel, then coordinates to update the dual variable.

V. Augmented Lagrangians

A. Definition of Strongly Convex Function

Definition: A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is **strongly convex** if the function *f* satisfy

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{m}{2} \|y - x\|_2^2$$

with $x, y \in \mathbb{R}^n$ for a certain $m > 0$, and $\|\cdot\|$ denotes a norm.

B. Augmented Lagrangians

Theorem (Hestenes, Powell 1969): We use the method of multipliers (MM) and the strongly convex function to simply swap the Lagrangian for an Augmented Lagrangian form:

$$L_\rho(x,\lambda) = f(x) + \lambda^T (Ax - b) + \frac{\rho}{2} \|Ax - b\|_2^2$$

for a certain penalty $\rho > 0$. Note that penalty ρ makes new objective strongly convex. This form leads to a better way to solve the optimization problem.

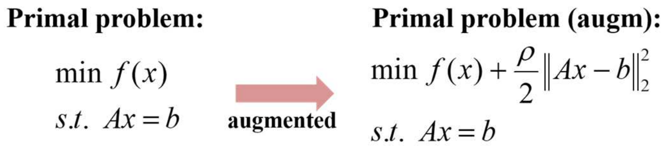


Figure 6. primal problem and primal problem(angmented).

Now, it can be iteratively computes until convergence:

$$\begin{cases} \lambda_k \quad = \lambda_{k-1} + \alpha_{k-1} \nabla g(\lambda_{k-1}) \\ x_{k+1} = arg\,\min_x L_\rho(x, \lambda_k) \end{cases},\ k = 1,2,\ldots$$

and leads to the DAM algorithm shown in Algorithm 2 below:

Algorithm 2. Augmented Lagrangian Method (ALM)

```
1: Initialize  dual varible $\lambda_0$and choose $\rho > 0$
2: repeat
3:     for each iteration do
4:         $x_{k+1} \leftarrow arg\,\min_x L_\rho(x_k, \lambda_k)$
5:         $\nabla g(\lambda) = Ax_{k+1} - b$
6:         $\lambda_{k+1} \leftarrow \lambda_k + \rho \nabla g(\lambda)$
7:     end for
8: until get $\lambda_{opt}$
```

VI. Alternating Direction Method of Multipliers (ADMM)

ADMM combines the advantages of DD and MM. It solves problems of the form:

$$\min f(x) + g(z)$$

$$\text{subject to}\ \ Ax + Bz = c$$

where *f* and *g* are both convex. Note that the objective is separable into two sets of variables. ADMM defines and uses a special augmented Lagrangian to allow for decomposition:

$$\begin{aligned} L_\rho(x,z,\lambda) = f(x) + g(z) + \lambda^T (Ax + Bz - c) \\ + \frac{\rho}{2} \|Ax + Bz - c\|_2^2 \end{aligned}$$

Now our iteration computes until convergence:

$$\begin{cases} \lambda_k \quad = \lambda_{k-1} + \alpha_{k-1} \nabla g(\lambda_{k-1}) \\ x_{k+1} = arg\,\min_x L_\rho(x, z_k, \lambda_k) \\ z_{k+1} = arg\,\min_z L_\rho(x_{k+1}, z, \lambda_k) \end{cases},\ k = 1,2,\ldots$$

At iteration *k*, we minimize for *x*, then *z*, and finally, update λ, keeping the other variables constant during each minimization. This gives us the ADMM algorithm shown in Algorithm 3:

Algorithm 3. ADMM

```
1: Initialize dual variable $\lambda_0$, $x_0$ and $z_0$
2: repeat
3:     for each iteration do
```

```
4:        $x_{k+1} \leftarrow arg \min_x L_\rho(x, z_k, \lambda_k)$
5:        $z_{k+1} \leftarrow arg \min_z L_\rho(x_{k+1}, z_k, \lambda_k)$
6:        $\nabla g(\lambda) = Ax_{k+1} + Bz_{k+1} - c$
6:        $\lambda_{k+1} \leftarrow \lambda_k + \rho\nabla g(\lambda)$
7:    end for
8: until get $\lambda_{\text{opt}}$
```

VII. IMPLEMENTATION

For illustration, an example for ADMM is given as follows:

$$min \;\; (x-1)^2 + (y-2)^2$$

$$\text{subject to } \begin{cases} 0 \le x \le 3, \\ 1 \le y \le 4, \\ 2x + 3y = 5 \end{cases}$$

This is a primal problem. Actually our objective can be thought as a distance between a point $P(x,y)$ and a point $A(1,2)$ and feasible region is a line segment. We have to find an optimal solution $P^*(x^*, y^*)$ to minimize the objective function.

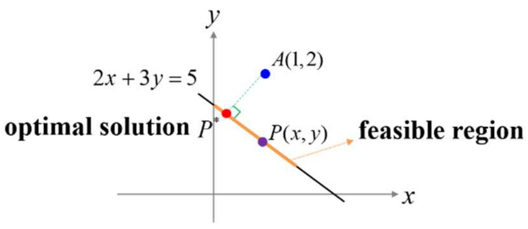


Figure 7. The primal problem.

ADMM defines and uses a special augmented Lagrangian to allow for decomposition:

$$L_\rho(x, y, \lambda) = (x-1)^2 + (y-2)^2 + \lambda(2x + 3y - 5) + \frac{\rho}{2}(2x + 3y - 5)^2$$

and

$$\nabla g(\lambda) = 2x + 3y - 5$$

Now, the iteration computes until convergence:

$$\begin{cases} \lambda_k \;\; = \lambda_{k-1} + \alpha_{k-1}\nabla g(\lambda_{k-1}) \\ x_{k+1} = arg \min_x L_\rho(x, y_k, \lambda_k) \\ y_{k+1} = arg \min_y L_\rho(x_{k+1}, y, \lambda_k) \end{cases}, k = 1,2, \ldots$$

At iteration k, we minimize for x, then y, and finally, update λ, keeping the other variables constant during each minimization. Here, the use of ADMM algorithm in Matlab to achieve an optimal solution $(x^*, y^*) \approx (0.53846, 1.3077)$ and the minimization $f(x^*, y^*) \approx 0.692$ for this example below:

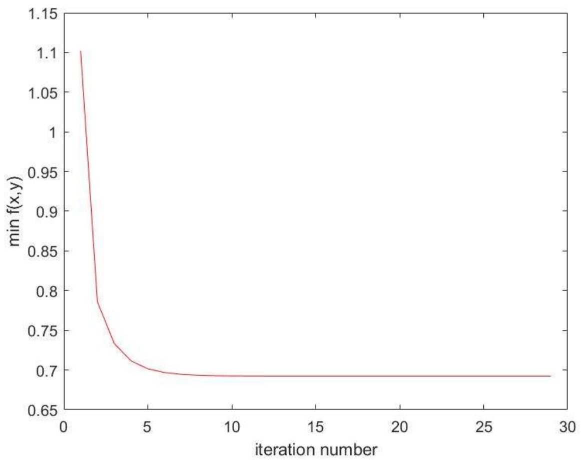


Figure 8. The convergence graph shows that at least 5 iterations to achieve minimization of $f(s, y)$.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support grants from Ministry of Science and Technology of Taiwan, MOST 110-2221-E-305-012, and University System of Taipei Joint Research Program, USTP-NTPU-TMU-110-02.

REFERENCES

[1] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, “Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers”, Foundations and Trends in Machine Learning, vol. 3, no.1, pp. 1-122, 2010.

[2] M. R. Hestenes, “Multiplier and gradient methods”, Journal of Optimization Theory and Applications, vol. 4, pp. 302-320, 1969.

[3] M. J. D. Powell, “A method for nonlinear constraints in minimization problems”, in Optimization, (R. Fletcher, ed.), Academic Press, 1969.

[4] Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de Dirichlet non linéares R. Glowinski and A. Marrocco, Revue Française d'Automatique, Informatique, et Recherche Opérationelle, 1975

[5] D. Gabay and B. Mercier, A dual algorithm for the solution of nonlinear variational problems via finite element approximations Computers and Mathematics with Applications, 1976

[6] A. Beck, First-Order Methods in Optimization (2017), chapter 5.

[7] Yu. Nesterov, Lectures on Convex Optimization (2018), section 2.1. (The result page 1.32 is Theorem 2.1.7 in the book.)

[8] B. T. Polyak, Introduction to Optimization (1987), section 1.4.

[9] The example on page 1.4 is from N. Z. Shor, Nondifferentiable Optimization and Polynomial Problems (1998), page 37.

[10] J.-B.Hiriart-Urruty,C. Lemaréchal, Convex Analysis and Minimization Algoritms (1993), chapter X.

[11] D.P. Bertsekas, A. Nedic, A.E. Ozdaglar, Convex Analysis and Optimization (2003), chapter 7.

[12] R. T. Rockafellar, Convex Analysis (1970).