Contents lists available at Science-Gate

International Journal of Advanced and Applied Sciences

Journal homepage: http://www.science-gate.com/IJAAS.html

A hyperbolic penalty method to solve structured convex minimization problems

Abdelouahed Hamdi [1], Temadher K. Al-Maadeed [1, *], Akram Taati [2]

[1]*Department of Mathematics, Statistics, and Physics, College of Arts and Sciences, Qatar University, Doha, Qatar*
[2]*Department of Mathematics, University of Guilan, Rasht, Iran*

A R T I C L E I N F O

Article history:
Received 1 December 2019
Received in revised form
15 February 2020
Accepted 20 February 2020

Keywords:
Hyperbolic penalty methods
Decomposition
Convex functions
Large scale optimization

A B S T R A C T

This paper presents a decomposition algorithm based on the smooth hyperbolic penalty, which leads to a scheme suitable for parallelized computations. The proposed algorithm can be seen as a separable version of the earlier hyperbolic penalty method built, and its main idea is related to a penalty-type scheme mixed with a kind of resource allocation approach to decompose large scale separable constrained minimization programs.

1. Introduction

There has been considerable recent interest in solving large-scale optimization problems. Such problems arise in many specialties, for instance, multistage stochastic optimization, distributed model predictive control, transportation, telecommunication models, networks, and deep learning. More recently, the technology of big and fast computers favorable to parallel computations has helped a lot and encouraged many researchers to continue tackling larger and larger models. But it is still a challenging area where many real-world engineering problems are waiting for more and more bright ideas (classical or non-classical ones) to cope with their multi-millions decision variables. Basic references are Bertsekas and Tsitsiklis (1989) and Lasdon (1970), where the main motivation for decomposing appear separately, mainly:

- Splitting into many weakly coupled subsystems.
- Partitioning variable and/or constraint sets to isolate easier subproblems; decentralizing global optimal decisions among local decision levels.
- Parallelizing or distributing computations on specific parallel computing device.

Depending on the motivation, the way to split or partition the model may rely on different structural properties of a given problem. It is well-known that the classical coordination functions are in general non-smooth and, therefore, hard to be optimized, turning the price to pay for decomposition too high for computational purposes. The computational cost is not the only drawback as the non-smoothness of the coordination function almost always implies that non-unique solutions are introduced in the subproblems, which reduce the possibility of decentralizing them completely. These are the reasons why some penalization ideas, regularization techniques like the proximal point method look attractive. Besides the possibility of treating non-smooth convex coordination problems efficiently, the introduction of the quadratic penalty terms, for instance, in the subproblems, can impose unique solutions to guarantee decentralized procedures. Classical primal penalty quadratic methods (Fiacco and McCormick, 1990; Fletcher, 1975; Kort and Bertsekas, 1976; Polyak, 2001), when applied to large separable constrained models, lose the separability which allows decomposition of the subproblems' penalized objective. Indeed, this is removed by the presence of quadratic terms in the penalized potential functional. The same diagnostic can be reached by applying many other penalty schemes. Particularly, the Hyperbolic penalty method introduced and studied by Xavier (2001) and used in Melo et al. (2011; 2012) and Evirgen (2017).

Motivated by the effective results obtained in Melo et al. (2011) to solve some difficult mathematical problems with complementarity constraints (MPCC) and those obtained in Evirgen (2017), where a nonlinear dynamic system was constructed using the hyperbolic penalty function

* Corresponding Author.
Email Address: abhamdi@qu.edu.qa (T. K. Al-Maadeed)
https://doi.org/10.21833/ijaas.2020.05.011
Corresponding author's ORCID profile:
https://orcid.org/0000-0003-1950-8907
2313-626X/© 2020 The Authors. Published by IASE.

for a certain class of inequality constrained optimization problems. We aim to use this intriguing hyperbolic penalty function in order to develop a decomposition scheme favorable to parallel computations. Polyak (2001) proposed a nonlinear re-scaling algorithm based on the log-sigmoid functional, and he obtained some nice results and properties. This again motivated us to re-launch the use of the new hyperbolic functional (Xavier, 2001), which is twice continuously differentiable and is combining features of both exterior and interior penalty methods. Our approach consists in mixing the hyperbolic penalty algorithmic scheme with some recent decomposition algorithms developed by Hamdi et al. (1997), Hamdi and Mahey (2000) and Hamdi (2005a, 2005b). These decomposition class of methods known as separable augmented Lagrangian algorithms (SALA) can be derived from the resource directive sub-problems associated with the coupling constraints. A complete review of decomposition methods for convex and non-convex optimization minimization problems can be found in Hamdi and Mishra (2011).

The paper is based upon the idea originally discussed by Hamdi et al. (1997), where the authors proposed a decomposable scheme that overcame the non-separability of the obtained augmented Lagrangian function. But here, we limit ourselves to propose a primal method based on the hyperbolic penalty functional far from augmented lagrangians where we have separable penalized subproblems.

This idea adds to the large number of publications that we find in the literature, aiming to build separable subproblems having strong legitimate theoretical properties. We limit ourselves to smooth convex cases, where some direct strategies have been proposed to exploit the inner structure of the penalty function and turn it into a separable one (see, for instance, the survey (Hamdi and Mishra, 2011) and references therein).

The paper is organized as follows: In the next section, we present the hyperbolic penalty method of Xavier, followed by the proposed decomposition method. Section 4 contains the convergence analysis of the new algorithms. The last section contains some algorithmic issues and suggestions to extend the proposed algorithm.

2. Hyperbolic penalty method

Let f be a convex real-valued function and let $\left(g_1(x),\cdots,g_p(x)\right)^{\mathrm{T}}$ be finite concave real-valued functions on R^n, and consider the convex programming problem:

$$\min_{x\in\Re^n}\{f(x)\ :\ g_i(x)\geq 0, i=\overline{1,m}\}(C).$$

Using the Penalty function

$$P(y,\alpha,\tau)=\frac{-\tan(\alpha)}{2}y+\sqrt{\left(\frac{\tan(\alpha)}{2}\right)^2 y^2+\tau^2}.$$

Alternatively, the hyperbolic penalty function may be put in a more convenient form:

$$P(y,\lambda,\tau)=-\lambda y+\sqrt{\lambda^2y^2+\tau^2},\qquad \lambda=\frac{\tan(\alpha)}{2},$$

where, $\lambda\geqslant 0$ and $\tau\geqslant 0$. The graphic representation of $P(y,\alpha,\tau)$, as shown in Fig. 1, is a hyperbola with two asymptotes, a slant one forming an angle $\pi-\alpha$ with the x-axis and a horizontal one. Also, the graph has τ as the y-intercept.

Here are some properties of the function $P(y,\alpha,\tau)$ that will be used in this paper. All these properties are proved in Xavier (2001).

Properties:

- $P(y,\lambda,\tau)$ is k-times continuously differentiable for any positive integer k for $\tau>0$.
- $P(y,\lambda,\tau)$ is asymptotically tangent to the straight lines $r_1(y)=-2\lambda y$ and $r_2(y)=0$ $for \tau>0.$
- $P(y,\lambda,0)=0$ for $y\geq 0$, $P(y,\lambda,0)=-2\lambda y$ for $y<0$.
- $P(y,\lambda,\tau)\geqslant -2\lambda y$, for all $y\in\Re$, $\lambda\geq 0,\tau\geq 0.$
- $P(0,\lambda,\tau)=\tau$ for $\tau\geq 0$ $and \lambda\geq 0.$
- $P(y,\lambda,\tau)$ is:
  - A convex decreasing function of y for $\tau>0$ and $\lambda\geq 0$.
  - A convex non-increasing function of y for $\tau=0$ and $\lambda\geq 0$.
  - A convex function equal to τ for $\lambda=0$.
- For $\lambda_{k+1}>\lambda_k$ and $\tau=0$
  - $P(y,\lambda_{k+1},\tau)<P(y,\lambda_k,\tau)$ for $y>0$
  - $P(y,\lambda_{k+1},\tau)=P(y,\lambda_k,\tau)$ for $y=0$
  - $P(y,\lambda_{k+1},\tau)>P(y,\lambda_k,\tau)$ for $y<0$.
- $P(y,\lambda,\tau_{k+1})<P(y,\lambda,\tau^k)$ for all $y\in\Re$, $\lambda>0$, $0\leq\tau_{k+1}<\tau_k$.
- $max_y\{P(y,\lambda,\tau^0)-P(y,\lambda,\tau^1)\}=\tau_0-\tau_1$, and it occurs at $y=0$ for $0\leq\tau_0<\tau_1$.

3. Hyperbolic decomposition algorithm (HDA)

3.1. Model formulation

In this section, we will build up a new hyperbolic decomposition algorithm (HDA) algorithm to solve too large scale convex inequality constrained programs with separable structure. We are concerned here with block separable nonlinear constrained optimization problems:

$$\min_{x\in\Re^n}\{F(x)=\textstyle\sum_{i=1}^p f_i(x_i): x\in\Omega,\}(SP),$$

where $f_i:\Re^{n_i}\to\ I\ -2.5pt\ R$ are all convex functions, and

$$\Omega=\left\{x\in\Re^n: \sum_{i=1}^p g_i(x_i)\geq 0,\qquad\right\}$$

is the convex set where g_i are defined from $\Re^{n_i} \to \Re$ for $i = \overline{1,p}, \sum_{i=1}^{p} n_i = n$.

The above constraint is usually referred to as a coupling constraint. Along with this work, all the functions f_i, g_i are C^2 and we assume the following assumptions.

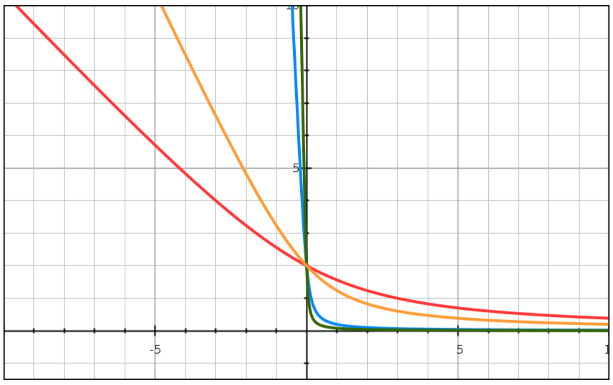

Fig. 1: $P(y,\lambda,2)$: Orange $\lambda = 0.5$, Red $\lambda = 1$, Blue $\lambda = 10$, Green $\lambda = 25$

Assumptions 1:

A1: The optimal set X^* of (SP) is nonempty and bounded.
A2: The Slater's condition holds, i.e.,

$$\exists \bar{x} \in \Re^n: \sum_{i=1}^{p} g_i(\bar{x}_i) > 0,\ j = \overline{1,m}.$$

3.2. New HDA

To build our new decomposition algorithm, we follow the well-known resources allocation scheme. Since we aim to develop an algorithm to decompose large scale, structured optimization models, we thought about the iterative approach studied and developed by Hamdi et al. (1997), Hamdi (2005b), and Hamdi and Mishra (2011), the Separable Augmented Lagrangian Algorithm (SALA). To this goal, p allocation vectors

$$y \in A = \{z \in \quad I \quad -2.5pt \ R^{\,p} \ |\sum_{i=1}^{p} z_i = 0\}$$

are added in a such a way to get the equivalent problem†,

$$(SPy)\begin{cases} \min & \sum_{i=1}^{p} f_i(x_i) \\ such\ that & g_i(x_i) + y_i \geq 0, \qquad i = \overline{1,p} \\ & y \in A \\ & x_i \in \quad I \quad -2.5pt \ R^{\,n_i}, \qquad i = \overline{1,p}, \end{cases}$$

to which we apply the Hyperbolic Penalty Method introduced previously with partial elimination of the constraints. In other words, for $\lambda > 0$, $\tau > 0$, the potential function related to the problem (SP_y) is defined as follows:

$$F(x,y,\lambda,\tau) = \sum_{i=1}^{p} f_i(x_i) \ + \ \sum_{i=1}^{p} P(g_i(x_i) + y_i, \lambda, \tau), \quad (1)$$

† If (x^*, y^*) is an optimal solution to SP_y then x^* is an optimal solution to (SP).

where $P(a,b,c) = -a\,b + \sqrt{a^2b^2 + c^2}$. Thus it is clear that the functional F is separable, i.e.,

$$F(x,y,\lambda,\tau) = \sum_{i=1}^{p} F_i(x_i, y_i, \lambda, \tau),$$

where,

$$F_i(x_i, y_i, \lambda, \tau) = f_i(x_i) + P(g_i(x_i) + y_i, \lambda, \tau).$$

Thus, the Hyperbolic Penalty algorithm can be applied as follows:

For any λ, $\tau > 0$, the following minimization problem,

$$(x^{k+1}, y^{k+1}) \to \min_{y \in A,\ \ x \in \Re^n} F(x, y, \lambda_k,\ \tau_k) \quad (2)$$

and

$$\lambda^{k+1} = r\,\lambda_k, \quad r > 1, \qquad \tau^{k+1} = q\,\tau_k, \quad 0 < q < 1. \quad (3)$$

The minimization in Eq. 2 is done by alternating the minimization with respect to x, then followed by the one with respect to the allocation variable y. i.e., we fix $y = y^k$ and find:

$$x^{k+1} \in Arg\min_x F(x, y^k, \lambda_k, \tau_k) = Arg\min_x \sum_{i=1}^{p} F_i(x_i, y_i^k, \lambda_k, \tau_k).$$

Then we can split the above minimization into p independent sub-problems with low-dimension. i.e.,

$$x_i^{k+1} \in Arg\min_{x_i}\{f_i(x_i) + P(g_i(x_i) + y_i^k, \lambda_k, \tau_k)\}$$

and now we fix $x = x^{k+1}$ to solve for y^{k+1}

$$y^{k+1} \in Arg\ min\{\sum_{i=1}^{p} P[g_i(x_i^{k+1}) + y_i, \lambda_k, \tau_k] : \sum_{i=1}^{p} y_i = 0\}. \quad (4)$$

It is not hard to solve the minimization explicitly with respect to y as shown in the following lemma, which gives us also an important remark about allocation variables.

Lemma 1: According to Eq. 4, y^{k+1} satisfy

$$y_i^{k+1} = -g_i(x_i^{k+1}) + \delta^{k+1}, \quad i = \overline{1,p}. \quad (5)$$

where $\delta^{k+1} = p^{-1} \sum_{i=1}^{p} g_i(x_i^{k+1})$.

Proof: By writing the classical Lagrangian to Eq. 4

$$L^k(y,t) = \sum_{i=1}^{p} P(g_i(x_i^{k+1}) + y_i, \lambda, \tau) + t\ \sum_{i=1}^{p} y_i$$

where $t \in \quad I \quad -2.5pt \ R$, and using the optimality, with the fact that $\sum_{i=1}^{p} y_i^{k+1} = 0$, we obtain after some direct calculations

$$t = \lambda - \frac{\lambda^2(g_i(x_i^{k+1}) + y_i^{k+1})}{\sqrt{\tau^2 + \lambda^2\,(g_i(x_i^{k+1}) + y_i^{k+1})^2}}, \quad (6)$$

which means that $\lambda - t$ does not depend on i and has the same sign as $g_i(x_i^{k+1}) + y_i^{k+1}$ for any $i = \overline{1,p}$. Now, according to Eq. 6, we have:

$$g_i(x_i^{k+1}) + y_i^{k+1} = \frac{\tau(\lambda - t)}{\lambda\sqrt{\lambda^2 - (\lambda - t)^2}}, \quad i = \overline{1,p}, \tag{7}$$

and straightforwardly after summing both sides and using the definition of δ^{k+1} we reach

$$(p\,\delta^{k+1})^2 = \frac{p^2\,\tau^2\,(\lambda - t)^2}{\lambda^2[\lambda^2 - (\lambda - t)^2]}, \tag{8}$$

and after some obvious calculations, we obtain:

$$(\lambda - t)^2 = \frac{\lambda^4 (p\,\delta^{k+1})^2}{\lambda^2 (p\,\delta^{k+1})^2 + p^2\tau^2}, \tag{9}$$

and by plugging directly Eq. 9 in Eq. 7 and after some simplifications and using the fact that $\lambda - t$ and δ^{k+1} has the same sign as $g_i(x_i^{k+1}) + y_i^{k+1}$, we obtain directly

$$y_i^{k+1} = -g_i(x_i^{k+1}) + \delta_{k+1}, \quad i = \overline{1,p}.$$

Algorithm 1:

```
Hyperbolic Decomposition Algorithm (HDA):
Step 1: Select $\tau_0 > 0$, $\lambda_0 > 0$, $y^0 = (y_1^0, \cdots, y_p^0)$, s.t $\sum_{i=1}^{p} y_i^0 = 0$, $\epsilon_1 > 0$, $\epsilon_2 > 0$,
        $r > 1$, $0 < q < 1$, $k = 0$.
Step 2: Determine : for any $i = \overline{1,p}$
        $x_i^{k+1} := arg \min_{x_i \in\ I\ -2.5pt\ R^{n_i}} \left\{ f_i(x_i) - \lambda_k (g_i(x_i) + y_i^k) + \sqrt{\tau_k^2 + \lambda_k^2 (g_i(x_i) + y_i^k)^2} \right\}$.
Step 3: Stoping criteria: Stop.
  Else : go to step 4
Step 4: Update and go back to step 2:
        $y_i^{k+1} = -g_i(x_i^{k+1}) + \delta^{k+1}$, $i = \overline{1,p}$, $\delta^{k+1} = \frac{1}{p}\sum_{i=1}^{p} g_i(x_i^{k+1})$
        $\lambda_{k+1} = r\,\lambda_k$, $\tau_{k+1} = q\,\tau_k$.
```

4. Properties and convergence analysis

For our analysis, we need some assumptions.

Assumptions 2:

A3: There is a pair (λ_0, τ_0) such that:

$$\inf_{y_i, x_i \in\ I\ -2.5pt\ R} F_i(x_i, y_i, \lambda_0, \tau_0) = F_i^0 > -\infty.$$

A4: There is a value $\epsilon > 0$ such that the set

$$\Omega_\epsilon = \left\{ x \in \Re^n : \sum_{i=1}^{p} g_i(x_i) > -\epsilon \right\}.$$

is bounded.

A5: The derivative of the functions f_i and g_i, $i = \overline{1,p}$ are bounded in the set Ω_ϵ.

The following theorem shows the existence of the minimum of the function $F(x, y, \lambda, \tau)$ and consequently for the functions F_i.

Proposition 1: If the conditions (A1)-(A5) hold, then there exists $\overline{\lambda} \geq \lambda^0$ such that

$$\inf_{x \in\ I\ -2.5pt\ R^n} F(x, y, \lambda, \tau) = \min_{x \in \Re^n} F(x, y, \lambda, \tau)$$

for all $\lambda \geqslant \overline{\lambda}$ and $0 \leq \tau \leq \tau_0$, $\sum_{i=1}^{p} y_i = 0$.

Proof: We preferred to add it in the Appendix A of the paper.

Proposition 2: If conditions (A1)-(A5) are satisfied, then there exists a value $\overline{\overline{\lambda}}$ such that for all $\lambda \geq \overline{\overline{\lambda}}$ and $0 \leq \tau \leq \tau_0$, the minimum point $x(y, \lambda, \tau)$ of the modified objective function $F(x, y, \lambda, \tau)$ is feasible.

Proof: From prop 1, for $\lambda \geq \overline{\overline{\lambda}}$ there is $\tilde{x} \in \Omega_\epsilon$ such that:

$$\tilde{x}_i \in \operatorname{argmin}_{x_i} F_i(x_i, y_i, \lambda, \tau).$$

From the first-order optimality condition

$$\frac{\partial}{\partial x_i} F_i(\tilde{x}_i, y_i, \lambda, \tau) = 0$$

hence,

$$\frac{d}{dx_i} f_i(\tilde{x}_i) - \frac{\partial}{\partial x_i} P(g_i(\tilde{x}_i) + y_i, \lambda, \tau) \cdot g'_i(\tilde{x}_i) = 0,$$

which implies

$$|\frac{d}{d\,x_i} f_i(\tilde{x}_i)| = \left|\frac{\partial}{\partial x_i} P(g_i(\tilde{x}_i) + y_i, \lambda, \tau)\right| |g'_i(\tilde{x}_i)|. \qquad (10)$$

From the properties of the hyperbolic penalty function P for $y < 0$,

$$\lim_{\lambda\to\infty} P(y, \lambda, \tau) = -\infty,$$

and since f'_i and g'_i are bounded on Ω_ϵ then there is $\bar{\bar{\lambda}} \geqslant \bar{\lambda}$ such that Eq. 10 will be impossible unless $g_i(\tilde{x}_i) + y_i \geq 0$, $\forall i = \overline{1,p}$, and hence $\sum_{i=1}^{p} g_i(\tilde{x}_i) \geq 0$, in other words, the optimal point $\tilde{x}$ will certainly be feasible.

The next theorem will be similar to the result given in Xavier (2001), which shows a conditional convergence of a feasible minimum point sequence.

Theorem 1: If conditions (A1)-(A5) are satisfied, and if $\lim_{k\to\infty}\tau^k = 0$ and $\lambda \geq \bar{\bar{\lambda}}$ then a convergent sub-sequence $\{x^k\} \to \tilde{x}$ will exist, and the limit of any of these sub-sequences is an optimal point.

Proof: For any $\lambda \geq \bar{\bar{\lambda}}$ the point x^k will be feasible and then for any point $x^* \in X^*$ we have

$$\sum_{i=1}^{p} f_i(x_i^*) \leqslant \sum_{i=1}^{p} f_i(x_i^k). \qquad (11)$$

On the other hand, x^k is a minimum point of F. Then,

$$F(x^k, y^k, \lambda, \tau) \leqslant F(x^*, y^k, \lambda, \tau), \qquad \lambda \geq \bar{\bar{\lambda}}.$$

Therefore,

$$\lim_{k\to\infty} F(x^k, y^k, \lambda, \tau) \leqslant \lim_{k\to\infty} F(x^*, y^k, \lambda, \tau)$$
$$\lim_{k\to\infty} \sum_{i=1}^{p} f_i(x_i^k) + \sum_{i=1}^{p} P(g_i(x_i^k) + y_i^k, \lambda, \tau) \leqslant$$
$$\lim_{k\to\infty} \sum_{i=1}^{p} f_i(x_i^*) + \sum_{i=1}^{p} P(g_i(x_i^*) + y_i^k, \lambda, \tau).$$

Since $\lim_{k\to\infty}\tau^k = 0$ and from (P_1),

$$\lim_{k\to\infty} \sum_{i=1}^{p} f_i(x_i^k) \leqslant \sum_{i=1}^{p} f_i(x_i^*)$$

by Eq. 11 we have:

$$\lim_{k\to\infty} \sum_{i=1}^{p} f_i(x_i^k) = \sum_{i=1}^{p} f_i(x_i^*).$$

Since Ω is compact set then there exists a sub-sequence of $\{x^k\}$ that will converge to $\tilde{x} \in X^*$.

5. Numerical study

This section is devoted to some numerical tests where we study the numerical behavior of the HDA. The study will tackle the feasibility, optimality, and stability of this method with respect to the parameters involved. Furthermore, an extension of the HDA, the Proximal Hyperbolic Decomposition Algorithm (PHDA), described below, is tested. This study is completed by a brief comparison involving HDA, PHDA, and the well-known strong CVX tool for some Convex Programming models developed by S.T. Boyd and M.C Grant from Stanford University (Grant and Boyd, 2020). CVX is a Matlab-based modeling system for constructing and solving some convex programs (CPs). CVX supports a number of standard problem types, including linear and quadratic programs, and it is mainly based on primal-dual interior-point techniques.

In this section, we present some computational tests on the performance of the two presented algorithm in this paper for solving convex separable problems of such form:

$$\min\{\sum_{i=1}^{p} c_i\, x_i \;\; : \;\sum_{i=1}^{p} a_i x_i^2 + b_i x_i + d \geq 0,\; x_i \in \Re^{n_i}\} \quad P(n,p)$$

where $\sum_{i=1}^{p} n_i = n$, and

$$\min\{a_1(x_1 - 0.5)^2 + \sum_{i=2}^{m} a_i(x_i + 1)^2 + \sum_{i=m+1}^{n} a_i(x_i - 1)^2 \colon\; g(x) \geq 0\} \quad Q(n,m,\rho)$$

where

$$g(x) = b_1(1 - x_1) + \sum_{i=2}^{m} b_i\, x_i^2 + \sum_{i=m+1}^{n} b_i(x_i - 1)^2,$$

and,

$$a_i = \rho - \frac{\rho^2 - 1}{\rho(n-1)}(i-1),$$
$$b_i = \rho + \frac{\rho^2 - 1}{n-1}(i-1), \quad i = 1, 2, \cdots, n.$$

These problems were generated randomly using MATLAB. Generator and the size of the considered problems vary from 10 to 1,000,000. The programs were written in MATLAB version R2018b.

To solve the unconstrained minimization problems in Step 2 of Algorithms (HDA) and (PHDA), we used "fminunc" function in MATLAB (Coleman et al., 1999), amending its default setting to apply the Quasi-Newton method. In addition, the CVX employs a primal-dual interior-point algorithm (Grant and Boyd, 2014; 2020).

5.1. Algorithmic considerations

In practice, the penalty parameters τ, λ play a fundamental role in the behavior and efficiency of the proposed algorithms of type HDA and in general for any algorithms based on penalization and or on augmented (modified) Lagrangian. These parameters can be used to reach some accepted feasibility of the iterates. In order to avoid the case where λ becomes too large, we have fixed an upper bound λ_{max}. We have used convenient stopping criteria, which are similar to those used by Breitfeld and Shanno (Breitfeld and Shanno, 1996). This 2-parameters penalization depends on two positive parameters λ and τ which proceeds as follows:

- In the initial phase of the process, λ increases, causing a significant increase of the penalty at infeasible points, while a reduction in penalty is observed for points inside the feasible region. This way, the search is directed to the feasible region since the goal is to minimize the penalty.
- From the moment that a feasible point is obtained, the second parameter τ decreases.

Now, we propose to update λ and τ according to the following scheme

$$\begin{cases} \lambda_{k+1}^i = r\,\lambda_k^i, & \tau_{k+1}^i = \tau_k^i \quad \text{if } \; g_i(x_i^{k+1}) < 0 \\ \lambda_{k+1}^i = \lambda_k^i, & \tau_{k+1}^i = q\,\tau_k^i \quad \text{otherwise,} \end{cases}$$

where $r > 1$ and $0 < q < 1$.

5.2. HDA Performances

To evaluate the performance of implementable versions of HDA, we have presented in Tables 1-2 some partial results at each iteration with the corresponding given data and parameters. We have gathered the feasibility and the variations in the objective function at each iteration. Table 3 gives iterations number, feasibility, the variations in the objective function, and the needed time of computations for different sizes of model $P(n,p)$. Furthermore, Table 2 presents the changes in the parameter τ at each iteration (Note that the parameter $\lambda = 10$ is fixed).

Table 1: HDA for $P(100,2)$, $\lambda_0 = 10$ and $\tau_0 = 1$

Iter	Feasibility	$\lvert F(x^{(k+1)}) - F(x^k)\rvert$
1	4.1936e-01	1.8552e+01
2	4.2075e-02	3.7145e-01
3	4.2092e-03	3.7156e-02
4	4.2438e-04	3.7119e-03
5	5.7679e-05	3.5766e-04
6	8.1032e-06	4.8575e-05
7	1.1651e-06	6.8266e-06
8	6.7084e-08	1.0781e-06
9	4.3774e-09	6.1283e-08
10	1.0977e-09	3.2056e-09

Table 2: Algorithm HDA for $P(500,500)$, $\lambda_0 = 10$ and $q = 0.5$

Iter	τ	λ	$\lvert F(x^{(k+1)}) - F(x^k)\rvert$	Feasibility
1	2.5	10	35.105	264.99
2	1.25	10	21.804	264.23
3	0.625	10	77.689	180.41
⋮	⋮	⋮	⋮	
17	3.8147e-05	10	0.0075591	0.014325
18	1.9073e-05	10	0.0037705	0.0072049
19	9.5367e-06	10	0.0018808	0.0036412
⋮	⋮	⋮	⋮	
36	7.276e-11	10	2.404e-08	4.4237e-08
37	3.638e-11	10	1.0975e-08	2.2281e-08
38	1.819e-11	10	5.3219e-09	1.1487e-08

Table 3: HDA for $P(n,p)$ with $\lambda_0 = 10$ and $\tau_0 = 1$

n	p	Iter	Feasibility	$\lvert F(x^{(k+1)}) - F(x^k)\rvert$	CPU Time
5000	250	46	6.4984e-09	6.7364e-09	25.1389
5000	5000	51	1.8737e-08	7.0277e-09	382.717
10000	250	45	1.1804e-08	8.4542e-09	33.5606
10000	1000	48	1.1603e-08	5.4056e-09	115.541
12500	2500	50	1.1785e-08	4.9536e-09	245.936
12500	2500	49	1.7593e-08	9.9312e-09	249.963
15000	1500	48	1.7505e-08	8.009e-09	167.658
15000	5000	51	1.56472-08	6.8158-e9	470.214
25000	5000	51	1.5386e-08	8.8085e-09	520.194
50000	10000	53	8.422e-09	7.0049e-09	1067.88
75000	25000	54	1.4206-e08	9.3169e-09	2131.75
150000	10000	52	1.566-e08	5.8644e-09	1081.3
200000	10000	52	1.7434e-08	6.8612e-09	1116.12
300000	15000	51	2.7308e-08	9.3423e-09	1149.3
500000	20000	55	5.9431e-09	3.5798e-09	2378.06

In Tables 4-8, we have studied the influence of the penalty parameters τ and λ. We have used different initial values of these parameters, and some conclusions could be drawn according to these tables. In general the case $\tau_0 = 0.01,\ 0.5,\ 1$ are better than $\tau_0 = 10$ and more, but one has to be careful because sometimes when λ_k is so large, feasibility is rapidly reached, but optimality does not follow at the same speed (Table 7).

Table 4: HDA for $P(n,p)$ with $\lambda_0 = 1$, $\tau_0 = 25$

n	p	r	q	Iter	CPU Time
100	10	2	0.05	9	0.78667
100	10	2	0.1	14	1.10219
100	10	2	0.005	8	0.805029
100	10	$\sqrt{10}$	0.8	95	2.96916
100	10	$\sqrt{10}$	0.05	11	0.981767
250	25	$\sqrt{10}$	0.8	95	7.61844
250	25	$\sqrt{10}$	0.05	11	1.66404
250	50	2	0.5	36	3.68528
2500	250	3	0.5	40	25.7556

Table 5: Time (in sec): $HDA \ \rightarrow \ \tau_0 = 100$

Pbs	$\lambda_0 = 1$	$\lambda_0 = 5$	$\lambda_0 = 10$	$\lambda_0 = 25$	$\lambda_0 = 50$
$P(100,100)$	0.3149	0.2802	0.2798	0.2123	0.2143
$P(250,250)$	0.5521	0.5828	0.5122	0.4598	0.4876
$P(500,500)$	9.72063	9.13909	9.1225	9.25871	10.3971
$P(1000,1000)$	32.7545	32.7647	32.8279	33.0139	34.4139
$P(2500,2500)$	274.599	272.765	260.295	249.205	251.668

Remark 1: We observed that in the case where λ_0 was chosen less than 1, the method may reach rapidly feasible points but stuck far from the optimal value, or it diverges. For instance, for P(500,5) with

$\tau_0 = 100$, (HDA) diverges when $\lambda_0 = 0.1$ and $\lambda_0 = 0.7$. In addition, for $\lambda_0 = 0.9$, the algorithm stopped after reaching a feasible point with $f_{obj} = -111.981$ while the optimal objective obtained by the software CVX was $f^*_{CVX} = -112.08$. But, when $\lambda = 1$, CVX beats HDA lightly in time, and we get: $f^*_{obj} = -112.08 = f^*_{CVX}$ in 1.80012 sec, where CVX needs 1.59872 sec. The same conclusions were drawn for the following case ($P(50,50)$, $\tau 0 = 10$) (Table 6).

$\lambda_0 = 0.7 \longrightarrow f^*_{HDA} = -19.7908,\quad f^*_{CVX} = -26.9991,\quad Time = 1.69784,$

and for $\lambda_0 = 1$, we get:

$f^*_{HDA} = -26.9987,\quad Time = 4.08835,$
$f^*_{CVX} = -26.9991,\quad Time = 1.69784.$

In Tables 8 and 9, we tested problems $Q(n,m,\rho)$ for different dimensions from 100 to 12500 variables. The obtained results show the efficiency of HDA with respect to the CPU time used to reach convergence.

One may observe that the values of λ_0 are not influencing the CPU-Time heavily. Except when the number of variables is more than 5000, we observed that when λ_0 is greater of equal than 100 HDA is faster than the other tested cases (Tables 8 and 9).

Table 6: Time (in sec): $HDA \ \rightarrow\ \tau_0 = 100$

Pbs	$\lambda_0 = 0.05$	$\lambda_0 = 0.01$	$\lambda_0 = 0.1$	$\lambda_0 = 0.5$
$P(500,5)$	Div	Div	Div	Div
Pbs	$\lambda_0 = 0.7$	$\lambda_0 = 0.9$	$\lambda_0 = 1$	$\lambda_0 = 1.5$
$P(500,5)$	Div	almost cv	cv (1.80012 sec)	cv (4.0392 sec)

Table 7: CPU Time: $HDA \ \rightarrow\ \tau_0 = 0.1$

Pbs	$\lambda_0 = 100$	$\lambda_0 = 250$	$\lambda_0 = 500$	$\lambda_0 = 1000$	$\lambda_0 = 2000$
$P(1000,1000$	div	div	div	div	div
$P(1500,1500$	21.8024	22.0253	21.3952	26.7093	24.8805
$P(2500,2500$	111.793	div	div	div	div

Table 8: Time (in sec): $HDA \ \rightarrow\ \tau_0 = 1$

Pbs	$\lambda_0 = 100$	$\lambda_0 = 250$	$\lambda_0 = 500$	$\lambda_0 = 1000$	$\lambda_0 = 2000$
$Q(100,1,1)$	0.294983	0.258932	0.252127	0.290235	0.264605
$Q(100,50,1)$	0.291264	0.247257	0.234184	0.262379	0.24013
$Q(500,150,1)$	0.914938	0.881906	0.905574	0.91685	0.959121
$Q(500,100,1/2)$	1.11858	1.1027	1.05881	1.13897	1.32782
$Q(1000,800,1/4)$	0.629116	0.592817	0.619653	0.64378	0.657695
$Q(5000,2,1)$	3.54268	3.65463	4.47767	4.16171	4.4505
$Q(8000,3500,1)$	4.8168	4.70235	4.54321	5.58043	5.0586
$Q(12500,5000,1)$	8.06193	6.75195	6.9635	7.0534	6.99595

Table 9: CPU Time for $HDA \ \rightarrow\ \tau_0 = 1$

Problems	$\lambda_0 = 1$	$\lambda_0 = 5$	$\lambda_0 = 10$	$\lambda_0 = 20$	$\lambda_0 = 30$	$\lambda_0 = 50$
$Q(100,50,1)$	0.978355	0.77753	0.708556	0.747519	0.792013	0.814627
$Q(500,150,1)$	1.27856	1.17787	1.14185	1.28701	1.33009	1.40368
$Q(500,100,1/2)$	1.81536	1.13002	1.83701	1.30857	1.60893	1.9679
$Q(1000,800,1)$	0.757987	0.617783	0.560985	0.542147	0.546702	0.5657
$Q(5000,1,1)$	3.51629	3.73427	3.74421	3.89665	4.29453	4.41673
$Q(8000,3500,1)$	10.2324	11.8915	5.95551	4.19212	4.83697	5.16421
$Q(12500,5000,1)$	13.7405	21.0406	11.5494	9.1961	8.82265	8.50312

In the following Tables 10 and 11, we show that sometimes, we may face the influence of other internal parameters. For instance, the factors $(q,\ \rho)$ to increase and decrease the λ and τ respectively. According to many tests, feasibility is not altered, but the obtained objective function values may be unstable. This is due to the fast feasibility that stops the penalization effect of the algorithm and may cause slow convergence or produce some jumps around the optimal solution. Table 12, shows the CPU Time needed to solve large scale problems reaching 10^6 variables.

5.3. Extension and comparison

In order to increase the stability of the HDA and enrich it with some nice properties, we propose here to mix the proximal point technique (Rockafellar, 1976) with our (HDA).

In other words, we add a quadratic term $\frac{1}{2c} \parallel x - x^k \parallel^2$ to the hyperbolic penalty function. Thus, it is an easy exercise to extend the convergence analysis in Section 4 for our proximal hyperbolic decomposition algorithm.

Table 10: HDA for $P(n,p)$ with $\lambda_0 = 10,\ \tau_0 = 1$

n	p	r	q	f_{obj}
50	50	$\sqrt{8}$	0.6	-17.9542
50	50	$\sqrt{12}$	0.1	-17.2152
50	50	$\sqrt{8}$	0.05	-16.7776

Table 11: HDA for $P(n,p)$ with $\lambda_0 = 10,\ \tau_0 = 1$

n	p	r	q	f_{obj}
100	100	$\sqrt{8}$	0.6	-40.5734
100	100	$\sqrt{12}$	0.1	-41.158
100	100	$\sqrt{8}$	0.05	-40.3244

Table 12: CPU Time: HDA for $P(n,p)$

n	p	CPU Time	Number of Variables per sub-problem
50000	100	671.596	500
60000	1,000	286.759	60
70000	1,000	430.534	70
90000	1,000	468.946	90
100000	1,000	547.154	100
1000000	5,000	4623.25	200

Algorithm 2:

```
$Proximal\ Hyperbolic\ Decomposition\ Algorithm\ (PHDA)$:
$Step\ 1$: $Select\ \tau_0 > 0,\ \lambda_0 > 0,\ y^0 = (y_1^0, \cdots, y_p^0),\ s.t\ \sum_{i=1}^{p} y_i^0 = 0,\ \epsilon_1 > 0,\ \epsilon_2 > 0,$
        $r > 1,\ 0 < q < 1,\ c > 0,\ k = 0.$

$Step\ 2$: $Determine$: $for\ any\ i = \overline{1,p}$
    $x_i^{k+1} := arg \min_{x_i \in \Re^{n_i}} \left\{ f_i(x_i) - \lambda_k\,(g_i(x_i) + y_i^k) + \sqrt{\tau_k^2 + \lambda_k^2\,(g_i(x_i) + y_i^k)^2} + \frac{1}{2c}(x_i - x_i^k)^2 \right\}.$

$Step\ 3$: $If\ v_1 \le \epsilon_1\ ,\ v_2 \le \epsilon_2,\ where$
    $v_1 = -\delta^{k+1},$
        $v_2 = |f(x^k) - f(x^{k+1})|.$
  $Stop.$
  $Else$: $go\ to\ step\ 4$

$Step\ 4$: $Update\ and\ go\ back\ to\ step\ 2$:
    $\begin{cases} y_i^{k+1} = -g_i(x_i^{k+1}) + \delta^{k+1},\ i = \overline{1,p}. \\ \lambda_{k+1} = r\,\lambda_k,\ \tau_{k+1} = q\,\tau_k. \end{cases}$
```

As it was done for HDA, to evaluate the performance of implementable versions of PHDA, Tables 13 and 14 present some partial results per iteration.

Table 13: Time (in sec): $PHDA \rightarrow \tau_0 = 1,\ c_0 = 10,\ ** \hookrightarrow loc.\ min$

Problems	$\lambda_0 = 100$	$\lambda_0 = 250$	$\lambda_0 = 500$	$\lambda_0 = 1000$	$\lambda_0 = 2000$
$Q(100,1,1)$	1.44611	1.5348	1.43017	1.40501	1.50096
$Q(100,50,1)$	1.46166	1.37706	1.45547	1.43519	1.38023
$Q(500,150,1)$	1.70518	1.69838	1.73651	2.114	1.88389
$Q(500,100,1/2)$	1.86011	2.51599	1.83019	2.23843	1.32782
$Q(1000,800,1/4)$	**	**	**	**	**
$Q(5000,1,1)$	22.1233	22.2789	18.0528	15.4161	20.5188
$Q(8000,3500,1)$	82.2186	100.744	131.566	192.106	297.834

Table 14: Comparison: $\lambda_0 = 0.5,\ \tau_0 = 0.001$

		HDA		PHDA	
n	p	$Fval$	Time	$Fval$	Time
7500	75	-2322.71	10.2137	-2320.62	34.0168
7500	25	-2045.8	21.3731	-2044.85	54.4529
5000	1000	-1293.92	38.2184	-1292.55	41.1541
2500	50	-672.162	8.05189	-672.393	10.8952
1500	50	-283.072	5.68878	-283.237	8.1505
1000	500	-327.637	35.5733	-327.95	38.2991
500	500	-221.349	30.1621	-221.847	32.6024
500	10	-148.462	2.18094	-148.318	3.81574
100	10	-18.2256	1.01365	-18.1856	2.02456

In Tables 13-18, we compared the three algorithms on the same set of problems with different dimensions. We focused on the role of different parameters that produce nice results.

Remark 2:

- CVX for the above problems got many problems with memory, and sometimes it reached only feasible solutions. For large dimensions, CVX generally did not give good results. But for dimensions less than 5000, CVX is very efficient and beats HDA and its proximal version, as shown in Table 17.
- PHDA efficiency is closely related to HDA, but by tuning the proximal parameter c, we observed that PHDA is faster than HDA for solving problems $Q(n, m, \rho)$ as it is shown in Table 18. Not easy to understand, the inverse happened for problems $P(n, p)$ (Table 14). In general, we expected more stability around the optimal solution, but we think that the hyperbolic penalty term is dominant during the calculations, and the proximal term is unable to play its role as it should be. This situation is related to the proximal parameter c which needs some careful tuning in order to avoid any perturbations.

6. Conclusion

In this paper, we were interested in the development of the Hyperbolic Decomposition Algorithm (HDA) favorable to parallel computations. The main challenge was to overcome the non-separability of the obtained hyperbolic penalty function. The algorithm converges under some classical conditions and the nice properties of the hyperbolic functional. We ended our paper by a numerical study, where we discussed the behavior of the (HDA), its parameters' influence. In addition, we proposed a regularized version of HDA, the PHDA and we compared it to (HDA) and to a very strong algorithm CVX (Grant and Boyd, 2014). Since the (HDA) is mainly based on a penalization, so the penalty parameter influence is well-known and is shown in the numerical part. (HDA) can be made faster if we could program it on parallel processors machines.

Table 15: Comparison of HDA with CVX for $\lambda_0 = 0.5, \tau_0 = 0.001$ and $p = 1$

	HDA		CVX		
n	$Fval$	Time	$Fval$	Time	$\lvert Fvalalg1 - FvalCVX\rvert$
100	-2.9856e+01	3.77	-3.0389e+01	2.47	5.3307e-01
200	-3.6411e+01	3.81	-3.6414e+01	1.93	3.4272e-03
500	-1.2000e+02	17.68	-1.2124e+02	4.76	1.2458e+00
800	-3.5445e+02	35.33	-3.5503e+02	8.90	5.7573e-01
1000	-5.8513e+02	47.09	-5.8579e+02	10.85	6.6386e-01
1500	-4.8653e+02	100.89	-4.8969e+02	24.78	3.1535e+00

Table 16: HDA for $P(n,p)$ with $\lambda_0 = 1,\ \tau_0 = 25$

n	p	r	q	Iter	HDA-CPU Time	CVX-CPU Time
100	10	2	0.05	9	0.78667	5.88306
100	10	2	0.1	14	1.10219	5.87564
100	10	2	0.005	8	0.805029	6.94072
100	10	$\sqrt{10}$	0.8	95	2.96916	4.27314
100	10	$\sqrt{10}$	0.05	11	0.981767	6.03685
250	25	$\sqrt{10}$	0.8	95	7.61844	6.07923
250	25	$\sqrt{10}$	0.05	11	1.66404	7.13811
250	50	2	0.5	36	3.68528	4.49324
2500	250	3	0.5	40	25.7556	14.2906

Table 17: Comparison of HDA with CVX for $\lambda_0 = 0.5, \tau_0 = 0.001$ and $p = 1$

	HDA		CVX		
n	$Fval$	Time	$Fval$	Time	Comments
7500	-2499.11	498.877	1524.58	45.07	CVX Div.
5000	-1095.96	209.074	3780.12	30.426	CVX Div.
2500	9.457	52.431	681.357	14.148	CVX Div.
1500	-631.604	19.5159	-631.604	10.3011	
100	-286.232	9.833	-286.232	7.187	
800	308.033	6.719	308.033	8.203	
500	-134.375	3.802	-134.375	5.602	

Table 18: HDA, MHDA, and CVX for $\lambda_0 = 100, \tau_0 = 0.1,\ c_o = 10$, and $p = 1$ Problems $Q(m,n)$ with $\rho = 1$

		HDA		CVX		PHDA	
n	m	Fval	Time	$Fval$	Time	$Fval$	Time
100	2	0.793871	6.58139	0.793871	15.8529	0.793871	2.80084
500	100	78.9603	0.876856	78.9603	7.51149	78.9603	1.20662
1000	250	209.408	2.19094	209.408	12.1679	209.408	1.95086
1500	750	661.848	5.38192	661.848	18.1677	661.848	3.591
2500	1000	892.215	16.1945	NaN	*	892.215	10.9551
5000	2500	2296.45	48.5921	NaN	*	2296.45	26.1938
7500	4000	3718.64	106.094	NaN	*	3718.64	69.8502
10000	5000	4672.2	190.278	*	*	4672.2	97.6716
12500	4000	3718.64	346.232	*	*	3718.64	233.032
20000	8000	7547.61	3209.84	*	*	7547.61	4256.93

We tested a problem with one million variables subject to 5000 constraints in less than 4625 sec (Table 12). According to our experience with these type of algorithms, we think that a primal-dual version of (HDA) will be more stable and gives better results. A forthcoming paper will be devoted to the study of the performance of the Primal-Dual Hyperbolic Decomposition Algorithm with some comparisons with some well-known algorithms in literature as the Alternating Direction Multipliers Method (ADMM) (Hamdi and Mishra, 2011) and the Separable Augmented Lagrangian Algorithm (SALA)(Hamdi, 2005b). Particular efforts should be made to understand the interaction between the three parameters within the Proximal version of HDA.

Also, since (SALA) showed efficiency to reach nice local extrema, we believe that HDA and PHDA can also be efficient when solving some non-convex large-scale optimization problems.

Appendix A: Sub-problems solvability

Proposition 3: If the conditions (A1)-(A5) hold, then there exists $\bar{\lambda} \geq \lambda^0$ such that:

$$\inf_{x\in\Re^n} F(x,y,\lambda,\tau) = \min_{x\in\Re^n} F(x,y,\lambda,\tau)$$

for all $\lambda \geq \bar{\lambda}$ and $0 \leqslant \tau \leq \tau_0,\ \sum_{i=1}^{p} y_i = 0$.

Proof: Let $\tau_1 \in \ \ I \ \ -2.5pt \ \ R$ such that $\tau_0 > \tau_1$. By using P_8 and $\lambda^0 \geq 0$

$$F_i(x_i,y_i,\lambda^0,\tau_0) - F_i(x_i,y_i,\lambda^0,\tau^1) = P(g_i(x_i)+y_i,\lambda^0,\tau_0) - P(g_i(x_i)+y_i,\lambda^0,\tau^1)$$
$$\leqslant \tau_0 - \tau_1 \qquad (12)$$
$$\leqslant \tau_0.$$

Using (A3) and Eq. 12, we get

$$F_i^0 \leqslant F_i(x_i,y_i,\lambda^0,\tau_0) \leq \tau_0 + F_i(x_i,y_i,\lambda_0,\tau),\ \ \forall\tau\in[0,\tau_0],$$

then

$$F_i^0 - \tau_0 \leqslant F_i(x_i,y_i,\lambda_0,\tau) = f_i(x_i) + P(g_i(x_i)+y_i,\lambda_0,\tau) \qquad (13)$$

where, $\tau \in [0,\tau_0]$. Now, let $z \in \Omega$ be a feasible point, and by condition (A1) and (A4), the set Ω is compact, and since f_i are all continuous, then they attain their maximum value on the set Ω. Hence,

$$F(z,y,\lambda,\tau) \ = \ \sum_{i=1}^{p} f_i(z_i) + \sum_{i=1}^{p} P(g_i(z_i)+y_i,\lambda,\tau)$$
$$\leqslant \ \sum_{i=1}^{p} f_i(z_i^0) + \sum_{i=1}^{p} P(g_i(z_i)+y_i,\lambda,\tau)$$

where $F(z^0) = \sum_{i=1}^{p} f_i(z_i^0)$ is the maximum value of F on Ω.

Let $y_i^z = -g_i(z_i) + \frac{1}{p}\sum_{i=1}^{p} g_i(z_i)$, then using P_1 and P_3, we have for $\lambda > 0$,

$$P(g_i(z_i)+y_i^z,\lambda,\tau) < P(g_i(z_i)+y_i^z,0,\tau)=\tau$$

for $\tau > 0$. For $\tau = 0$, $P(g_i(z_i)+y_i^z,\lambda,\tau)=0=\tau$ since $g_i(z_i)+y_i^z > 0$.

Hence,

$$F(z,y^z,\lambda,\tau) \leqslant \sum_{i=1}^{p} f_i(z_i^0)+\tau p. \tag{14}$$

On the other hand, if w is not in the set Ω_ϵ, i.e., $\sum_{i=1}^{p} g_i(w_i) \le -\epsilon$. Then:

$$F_i(w_i,y_i,\lambda,\tau)=f_i(w_i)+P(g_i(w_i)+y_i,\lambda,\tau).$$

If $g_i(w_i)+y_i \ge 0$ then $P(g_i(w_i)+y_i,\lambda,\tau)>0$ and we conclude:

$$F_i(w_i,y_i,\lambda,\tau) > f_i(w_i). \tag{15}$$

Also, if $g_i(w_i)+y_i<0$, then from P_1 and P_4, we get

$$P(g_i(w_i)+y_i,\lambda,\tau) > -2\lambda(g_i(w_i)+y_i),$$

and thus,

$$F_i(w_i,y_i,\lambda,\tau) > f_i(w_i)-2\lambda(g_i(w_i)+y_i). \tag{16}$$

For the other side, using P_3, if $g_i(w_i)+y_i \ge 0$, we have:

$$P(g_i(w_i)+y_i,\lambda,\tau) < P(g_i(w_i)+y_i,0,\tau)=\tau$$

and if $g_i(w_i)+y_i<0$, it is easy to show that:

$$P(g_i(w_i)+y_i,\lambda,\tau) \le -2\lambda(g_i(w_i)+y_i)+\tau.$$

So, for $0 \leqslant \tau \le \tau_0$

$$F_i(w_i,y_i,\lambda_0,\tau)=f_i(w_i)+P(g_i(w_i)+y_i,\lambda_0,\tau) \leqslant f_i(w_i)+ \begin{cases} \tau_0 & if \quad g_i(w_i)+y_i \geqslant 0 \\ -2\lambda_0(g_i(w_i)+y_i)+\tau_0 & if \quad g_i(w_i)+y_i<0. \end{cases} \tag{17}$$

Therefore, using the above results:

- If $g_i(w_i)+y_i \ge 0$ from Eq. 15 to Eq. 17

$$F_i(w_i,y_i,\lambda,\tau) \geqslant F_i^0-2\tau_0-2(\lambda-\lambda_0)(g_i(w_i)+y_i). \tag{18}$$

- If $g_i(w_i)+y_i<0$ then from Eq. 16 and Eq. 18

$$\begin{aligned} F_i(w_i,y_i,\lambda,\tau) &\geqslant f_i(w_i)-2\lambda(g_i(w_i)+y_i). \\ &\geqslant f_i(w_i)-2\lambda(g_i(w_i)+y_i)-2\lambda_0(g_i(w_i)+y_i)+2\lambda_0(g_i(w_i)+y_i) \\ &\ge f_i(w_i)-2\lambda_0(g_i(w_i)+y_i)+(2\lambda_0-2\lambda)(g_i(w_i)+y_i) \\ &\geqslant F_i(w_i,y_i,\lambda_0,\tau)-\tau_0+(2\lambda_0-2\lambda)(g_i(w_i)+y_i) \end{aligned}$$

Using Eq. 13, we have:

$$F_i(w_i,y_i,\lambda,\tau) \geqslant F_i^0-2\tau_0-2(\lambda-\lambda_0)(g_i(w_i)+y_i). \tag{19}$$

In general, using Eq. 17 and Eq. 18

$$F_i(w_i,y_i,\lambda,\tau) \ge F_i^0-2\tau_0-2(\lambda-\lambda_0)(g_i(w_i)+y_i)$$

and hence,

$$F(w,y,\lambda,\tau)=\sum_{i=1}^{p} F_i(w_i,y_i,\lambda,\tau) \ge \sum_{i=1}^{p} F_i^0-2\tau_0 p-2(\lambda-\lambda_0)\sum_{i=1}^{p} g_i(w_i).$$

Using the fact that w is not in Ω_ϵ we have:

$$F(w,y,\lambda,\tau) \ge \sum_{i=1}^{p} F_i^0-2\tau_0 p+2(\lambda-\lambda_0)\epsilon.$$

Notice that the right side of the above inequality is an increasing function for λ. Therefore, there is a value $\bar{\lambda}$ such that this expression will be greater than $\sum_{i=1}^{p} f_i(z_i^0)+\tau_0 p$ for all $\lambda \ge \bar{\lambda}$ and $0 \le \tau \le \tau_0$, where

$$\bar{\lambda}=\lambda_0+\frac{\sum_{i=1}^{p}(f_i(z_i^0)-F_i^0)+3n\tau_0}{2\epsilon}.$$

Thus, for $\lambda \ge \bar{\lambda}$, we have from Eq. 14,

$$F(w,y,\lambda,\tau) \ge F(z,y^z,\lambda,\tau),$$

which implies that the $\inf_{x\in\ I\ -2.5pt\ R^n} F(x,y,\lambda,\tau)$ does not occur on the set $I\ -2.5pt\ R^n\backslash\Omega_\epsilon$, then

$$\inf_{x\in\ I\ -2.5pt\ R^n} F(x,y,\lambda,\tau)=\inf_{x\in\Omega_\epsilon} F(x,y,\lambda,\tau)=\min_{x\in\Omega_\epsilon} F(x,y,\lambda,\tau)$$

since Ω_ϵ is a compact set. The authors would like to thank the referees for their valuable remarks and suggestions to improve the final version of the paper.

Compliance with ethical standards

Conflict of interest

The authors declare that they have no conflict of interest.

References

Bertsekas DP and Tsitsiklis JN (1989). Parallel and distributed computation: Numerical methods. Volume 23, Prentice-Hall, Englewood Cliffs, USA.

Breitfeld MG and Shanno DF (1996). Computational experience with penalty-barrier methods for nonlinear programming.

Annals of Operations Research, 62(1): 439-463. https://doi.org/10.1007/BF02206826

Coleman T, Branch M, and Grace A (1999). Optimization toolbox for use with MATLAB: Users guide. The MathWorks Inc., Natick, Massachusetts, USA.

Evirgen F (2017). Solution of a class of optimization problems based on hyperbolic penalty dynamic framework. Acta Physica Polonica A, 132: 1062-1065. https://doi.org/10.12693/APhysPolA.132.1062

Fiacco AV and McCormick GP (1990). Nonlinear programming: Sequential unconstrained minimization techniques. Volume 4, SIAM, Philadelphia, USA. https://doi.org/10.1137/1.9781611971316

Fletcher R (1975). An ideal penalty function for constrained optimization. IMA Journal of Applied Mathematics, 15(3): 319-342. https://doi.org/10.1093/imamat/15.3.319

Grant M and Boyd S (2014). CVX: Matlab software for disciplined convex programming. Version 2.1. Available online at: https://bit.ly/2WzKf7U

Grant MC and Boyd SP (2020). The CVX users guide. CVX research Inc., Cambridge, UK. Available online at: https://bit.ly/3dj7RUh

Hamdi A (2005a). Two-level primal–dual proximal decomposition technique to solve large scale optimization problems. Applied Mathematics and Computation, 160(3): 921-938. https://doi.org/10.1016/j.amc.2003.11.040

Hamdi A (2005b). Decomposition for structured convex programs with smooth multiplier methods. Applied Mathematics and Computation, 169(1): 218-241. https://doi.org/10.1016/j.amc.2004.10.079

Hamdi A and Mahey P (2000). Separable diagonalized multiplier method for decomposing nonlinear programs. Computational and Applied Mathematics, 19(1): 1-29.

Hamdi A and Mishra SK (2011). Decomposition methods based on augmented Lagrangians: A survey. In: Mishra S (Eds.), Topics in nonconvex optimization: 175-203. Volume 50, Springer, New York, USA. https://doi.org/10.1007/978-1-4419-9640-4_11

Hamdi A, Mahey P, and Dussault JP (1997). A new decomposition method in nonconvex programming via a separable augmented Lagrangian. In: Gritzmann P, Horst R, Sachs E, and Tichatschke R (Eds), Recent advances in optimization: 90-104. Volume 452, Springer, Berlin, Germany. https://doi.org/10.1007/978-3-642-59073-3_7

Kort BW and Bertsekas DP (1976). Combined primal–dual and penalty methods for convex programming. SIAM Journal on Control and Optimization, 14(2): 268-294. https://doi.org/10.1137/0314020

Lasdon LS (1970). Optimization theory for large system. MacMillan, New York, USA.

Melo T, Monteiro MTT, and Matias J (2011). Solving MPCC problem with the hyperbolic penalty function. In the AIP Conference Proceedings, American Institute of Physics, 1389: 763-766. https://doi.org/10.1063/1.3636844

Melo T, Monteiro MTT, and Matias J (2012). Solving a signalized traffic intersection problem with a hyperbolic penalty function. In the AIP Conference Proceedings, American Institute of Physics, 1479 (1): 830-833. https://doi.org/10.1063/1.4756266

Polyak RA (2001). Log-Sigmoid multipliers method in constrained optimization. Annals of Operations Research, 101(1-4): 427-460. https://doi.org/10.1023/A:1010938423538

Rockafellar RT (1970). Convex analysis. Princeton University Press, Princeton, USA. https://doi.org/10.1515/9781400873173

Rockafellar RT (1976). Augmented Lagrangians and applications of the proximal point algorithm in convex programming. Mathematics of Operations Research, 1(2): 97-116. https://doi.org/10.1287/moor.1.2.97

Xavier AE (2001). Hyperbolic penalty: A new method for nonlinear programming with inequalities. International Transactions in Operational Research, 8(6): 659-671. https://doi.org/10.1111/1475-3995.t01-1-00330