Optimisation en Grande Dimension

César Arnould - Victor Bertret - Antoine Gicquel - Hugo Tessier

4GM | INSA Rennes

Année Universitaire 2020-2021

Table des matières

Introduction

L'optimisation en grande dimension vise à traiter des systèmes complexes, avec un grand nombre de variables et de contraintes, et qui requièrent beaucoup de calculs et de stockage d'informations. Durant ce projet, nous appliquerons les méthodes de décomposition/coordination à plusieurs problèmes d'optimisation liés à des problématiques de distribution d'eau, de portefeuille d'actions et de distribution d'énergie.

Les objectifs de ce projet sont multiples. Il vise tout d'abord à implémenter plusieurs méthodes de décomposition/coordination et à les comparer entre elles tant de manière théorique que par leurs efficacités numériques. Il a aussi pour objectif d'appliquer ces principes à des problématiques variées et de mettre en valeur les résultats obtenus.

Algorithmes de recherche d'un minimum

Tout au long de ce projet, nous nous intéresserons à des problèmes d'optimisation de la forme :

$$\begin{aligned} \min_{u \in \mathbb{R}^n} \quad & \langle Au, u \rangle - \langle b, u \rangle \\ \text{subject to} \quad & C_{eq} u = d_{eq} \\ & C_{in} u \leq d_{in} \end{aligned} \tag{$*$}$$

Dans tous les cas traités qui vont suivre, la matrice A sera symétrique et définie positive. La fonction objectif du problème $J(u) = \langle Au, u \rangle - \langle b, u \rangle$ sera donc (strictement) convexe. De plus, les contraintes d'égalité et d'inégalité étant linéaires, le problème est aussi lui-même convexe.
Afin de résoudre ce genre de problème, nous utiliserons plusieurs méthodes que nous avons implémenté de manière générique dans MATLAB.

Algorithme d'Uzawa

L'algorithme d'Uzawa est un algorithme de recherche de point selle. Il consiste à chaque itération à minimiser le Lagrangien en u avec les valeurs des multiplicateurs fixées puis de mettre à jour les valeurs des multiplicateurs par des opérations de projection ainsi qu'un pas de gradient (c'est un algorithme de gradient à pas fixe). Voici son adaptation permettant de résoudre des problèmes quadratiques ayant la forme présentée ci-dessus.

Algorithme 1 : Algorithme d'Uzawa pour le problème $(*)$

Initialisation des multiplicateurs λ et μ ;
$k = 1$;
tant que $k \leq 2$ *ou* $\left\|u^k - u^{k-1}\right\| / \left\|u^k\right\| > \epsilon$ **faire**
    Annulation du gradient du Lagrangien : $u^k = (2A)^{-1}(b - C_{in}^\top \mu - C_{eq}^\top \lambda)$;
    Mise à jour des multiplicateurs : $\begin{cases} \mu = \max\left(0, \mu + \rho(C_{in} * u - d_{in})\right) \\ \lambda = \lambda + \rho(C_{eq} * u - d_{eq}) \end{cases}$;
    $k = k + 1$;
fin

Algorithme d'Arrow-Hurwicz

L'algorithme d'Arrow-Hurwicz est également un algorithme de recherche de point selle. C'est une variante de l'algorithme d'Uzawa qui consiste à effectuer également un pas de gradient sur la variable u plutôt que d'effectuer la minimisation complète du Lagrangien. L'avantage de cette méthode par rapport à l'algorithme d'Uzawa est qu'elle ne nécessite pas de recherche explicite de minimum à chaque étape (et donc dans notre cas une inversion de matrice). Cependant, cette méthode possède 2 pas de gradient dont il faut optimiser les valeurs. Voici son adaptation permettant de résoudre des problèmes quadratiques ayant la forme présentée plus haut.

Algorithme 2 : Algorithme d'Arrow-Hurwicz pour le problème $(*)$

Initialisation de u^0 et des multiplicateurs λ et μ ;
$k = 1$;
tant que $k \leq 2$ *ou* $\left\|u^k - u^{k-1}\right\| / \left\|u^k\right\| > \epsilon$ **faire**
    Calcul de u^k (formule de projection) : $u^k = u^{k-1} - \alpha\left(2Au - b + C_{eq}^\top \lambda + C_{in}^\top \mu\right)$;
    Mise à jour des multiplicateurs : $\begin{cases} \mu = \max\left(0, \mu + \rho(C_{in} * u - d_{in})\right) \\ \lambda = \lambda + \rho(C_{eq} * u - d_{eq}) \end{cases}$;
    $k = k + 1$;
fin

Méthode de points intérieurs

Enfin, nous utiliserons un solveur intégré dans MATLAB permettant de résoudre des problèmes d'optimisation linéaires et des problèmes d'optimisation non linéaires convexes (comme le problème quadratique présenté plus haut) à l'aide de méthodes de points intérieurs.

Exercice 1 : Comparaison des algorithmes

On considère le problème d'optimisation suivant :

$$\begin{aligned} \min_{u\in\mathbb{R}^2} \quad & J(u) = \frac{1}{2}\langle Au, u\rangle - \langle b, u\rangle \\ \text{subject to} \quad & u_i + 2u_{i+1} \leq 0 \qquad \forall i \in \{1, \dots, N-1\} \\ & u_N \leq 0 \end{aligned} \tag{1}$$

On peut réécrire les contraintes sous la forme $Cu \leq 0$ avec $C := \begin{pmatrix} 1 & 2 & & (0) \\ & 1 & \ddots & \\ & & \ddots & 2 \\ (0) & & & 1 \end{pmatrix}$

On définit les variables du problème

- $u := (u_i)_{1\leq i\leq N} \in \mathbb{R}^N$
- $A := I_N$
- $b := (1, -1, 1, -1, \dots)^\top$

1.1 Représentation de l'ensemble des solutions admissibles

Dans cette première question, on va chercher à représenter graphiquement l'ensemble des solutions admissibles du problème, pour $N = 2$.

L'ensemble admissible K pour le vecteur u est $\mathbb{R}^2$, sous les contraintes $\begin{cases} u_1 + 2u_2 & \leq 0 \\ u_2 & \leq 0 \end{cases}$.

On représente cet ensemble sur MATLAB, en traçant en abscisse et ordonnée u_1 et u_2 respectivement. On représente également la solution optimale et les lignes de niveau correspondant à la projection de la fonction objectif J sur le plan. On représente également en 3 dimensions ce même ensemble admissible, la troisième dimension représentant les valeurs prises par J (qui forment une parabole d'équation $\frac{1}{2}(u_1^2 + u_2^2) + (u_1 - u_2) = 0$).

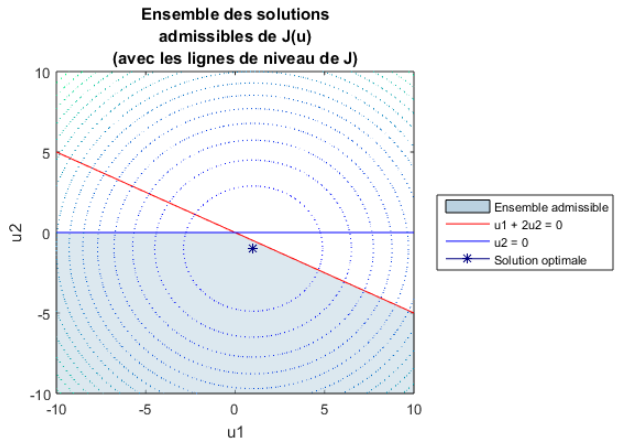

FIGURE 1 – Ensemble admissible 2D

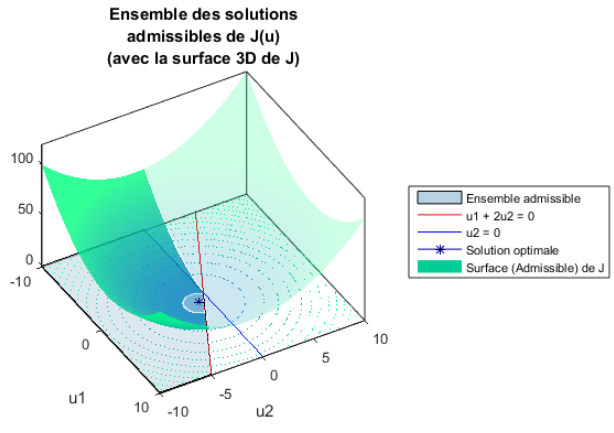

FIGURE 2 – Ensemble admissible 3D

On voit graphiquement qu'on a une infinité de solutions admissibles pour ces contraintes ; toutefois, la solution optimale u^* est proche de l'origine et ne sera pas trop difficile à calculer. Notons que le minimum de la fonction J sur l'ensemble admissible coïncide avec son minimum global (sans tenir compte des contraintes). On a ainsi $\min_{u\in K} J(u) = \min_{u\in\mathbb{R}^2} J(u)$.

1.2 Décomposition du problème

On remarque que ce problème possède une structure additive, on peut ainsi lui appliquer des méthodes de décomposition/coordination. Nous l'écrivons donc sous sa forme décomposable.

Écriture du problème (1) sous sa forme décomposable

On décompose l'objectif :

$$J(u) = \sum_{i=1}^{N} \frac{1}{2}u_i^2 + (-1)^i u_i = \sum_{i=1}^{N} J_i(u_i)$$

On décompose les contraintes :

$$\begin{pmatrix} 1 & 2 & & (0) \\ & 1 & \ddots & \\ & & \ddots & 2 \\ (0) & & & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix} \leq 0 \iff \begin{pmatrix} 1 \\ 0 \\ \vdots \\ \\ \end{pmatrix} u_1 + \begin{pmatrix} 2 \\ 1 \\ 0 \\ \vdots \\ \\ \end{pmatrix} u_2 + \cdots + \begin{pmatrix} \vdots \\ 0 \\ 2 \\ 1 \end{pmatrix} u_N \leq 0$$

$$\iff \sum_{i=1}^{N} \theta_i \, u_i \leq 0$$

Le problème s'écrit donc

$$\begin{array}{ll} \min\limits_{u\in\mathbb{R}^N} & \sum\limits_{i=1}^{N} J_i(u_i) \\ \text{s.t} & \sum\limits_{i=1}^{N} \theta_i \, u_i \leq 0 \end{array} \qquad \text{avec} \begin{cases} J_i(u_i) = \frac{1}{2}u_i^2 + (-1)^i u_i, & \forall i \in \{1,\dots,N\} \\ \theta_1 = e_1 & \\ \theta_i = 2e_{i-1} + e_i & \forall i \in \{2,\dots,N\} \end{cases}$$

On s'intéresse maintenant aux méthodes de décomposition par les prix, par les quantités et par prédiction et aux algorithmes qui leurs sont associés.

1.2.1 Décomposition par les prix

Décomposition Les sous-problèmes s'écrivent

$$\forall i = 1,\dots,N \qquad \min_{u_i\in\mathbb{R}} J_i(u_i) + \langle p^{(k)}, \theta_i u_i \rangle \tag{$\mathbb{P}_i^k$}$$

Coordination On actualise les prix pour des contraintes d'inégalité

$$p^{(k+1)} = \max\left(0, p^{(k)} + \rho_k C u^{k+1}\right)$$

avec $\rho_k > 0$ le pas de gradient dépendant du problème considéré.

On obtient le pseudo-code suivant pour l'algorithme de décomposition par les prix :

Algorithme 3 : Algorithme de décomposition par les prix

```
Initialisation des prix p^(1) ;
k = 1 ;
tant que k ≤ 2 ou ||u^k − u^{k−1}|| / ||u^k|| > ε faire
    Résolution des sous-problèmes (P_i^k) : calcul des u^{k+1};
    Coordination : calcul des p^(k+1) ;
    k = k + 1 ;
fin
```

1.2.2 Décomposition par les quantités

Décomposition Les sous-problèmes s'écrivent

$$\forall i = 1, \dots, N \quad \begin{cases} \min\limits_{u_i \in \mathbb{R}} & J_i(u_i) \\ \text{s.t} & \theta_i \, u_i \leq \omega_i^{(k)} \end{cases} \tag{$\mathbb{Q}_i^k$}$$

avec $\omega_i^{(k)} = (\omega_{i,1}^{(k)}, \dots, \omega_{i,N}^{(k)})^\top \in \mathbb{R}^N$ le vecteur des quantités tel que $\sum_{i=1}^N \omega_i^{(k)} = 0$.
On récupère les multiplicateurs de Lagrange $\lambda_i^{(k+1)}$ lors de la résolution de chaque sous-problème.

Coordination On actualise les quantités allouées à chaque sous-problème

$$\omega_i^{(k+1)} = \omega_i^{(k)} + \varepsilon_k \left(\lambda_i^{(k+1)} - \frac{1}{N} \sum_{i=1}^N \lambda_j^{(k+1)} \right)$$

On obtient le pseudo-code suivant pour l'algorithme de décomposition par les quantités :

Algorithme 4 : Algorithme de décomposition par les quantités

```
Initialisation des quantités ω_i^(1) = 0 ;
k = 1 ;
tant que k ≤ 2 ou ||u^k − u^{k−1}|| / ||u^k|| > ε faire
    Résolution des sous-problèmes (Q_i^k) : calcul des u^{k+1} et des λ_i^(k+1) ;
    Coordination : calcul des ω_i^(k+1) ;
    k = k + 1 ;
fin
```

1.2.3 Décomposition par prédiction

Résolution du sous-problème N

$$\begin{cases} \min\limits_{u_N \in \mathbb{R}} & J_N(u_N) \\ \text{s.t} & \theta_N \, u_N - v^{(k)} \leq 0 \end{cases} \tag{$\mathbb{R}_N^k$}$$

On récupère les multiplicateurs de Lagrange $\lambda^{(k+1)} = (\lambda_1^{(k+1)}, \dots, \lambda_N^{(k+1)})^\top$.

Calcul des prix On réalise une relaxation pour l'actualisation des prix

$$p^{(k+1)} = (1-\beta)p^{(k)} + \beta\lambda^{(k+1)}$$

Résolution des sous-problèmes restants

$$\forall i = 1, \dots, N-1 \qquad \min_{u_i \in \mathbb{R}} J_i(u_i) + \langle p^{(k)}, \theta_i u_i \rangle \qquad (\mathbb{R}_i^k)$$

Calcul de l'allocation On réalise une relaxation pour l'actualisation de l'allocation

$$v^{(k+1)} = (1-\gamma)v^{(k)} - \gamma \sum_{i=1}^{N-1} \theta_i u_i^{k+1}$$

On obtient le pseudo-code suivant pour l'algorithme de décomposition par prédiction :

Algorithme 5 : Algorithme de décomposition par prédiction

```
Initialisation de l'allocation v^(1) = 0 ;
Initialisation des prix p^(1) = 0 ;
k = 1 ;
tant que k ≤ 2 ou ||u^k − u^{k−1}|| / ||u^k|| > ε faire
    Résolution du sous-problème (ℝ_N^k) : calcul de u_N^{k+1} et des λ_N^{(k+1)} ;
    Coordination : calcul du prix p^(k+1);
    Résolution des sous-problèmes restants (ℝ_i^k) : calcul des u_i^{k+1} ;
    Coordination : calcul des v_i^(k+1) ;
    k = k + 1 ;
fin
```

1.2.4 Implémentation et résultats

D'un point de vue théorique, chaque méthode (décomposition par les prix, par allocations ou par prédiction) a ses avantages et ses inconvénients. La méthodes de décomposition par les prix a l'avantage d'être plutôt simple à implémenter (Lagrangien à calculer et Uzawa à appliquer), et conduit à des sous-problèmes bien formulés ; mais il faut attendre la convergence pour avoir la contrainte couplante satisfaite.

La méthode de décomposition par allocation de ressources (ou quantités), au contraire, donne une solution admissible à chaque itération mais elle est plus compliquée à implémenter que celle par les prix. Cette méthode présente également le risque de s'arrêter à une solution qui n'est pas optimale globale ou se bloquer en formulant des sous-problèmes définis sur l'ensemble vide.

Enfin, la méthode de décomposition par prédiction se veut un savant mélange des deux précédentes, ce qui la rend particulièrement complexe mais lui permet (souvent) de combiner les avantages des deux méthodes précédentes.

Le Lagrangien du problème s'écrit sous la forme :

$$\begin{aligned} L(u,p) &= \sum_{i=1}^{N} J_i(u_i) \; + \; \langle p, \sum_{i=1}^{N} \Theta_i(u_i) - \theta \rangle \\ &= \sum_{i=1}^{N} \frac{1}{2}\, u_i^2 + (-1)^i \, u_i \; + \; \langle p, \sum_{i=1}^{N} \theta_i \, u_i \rangle \end{aligned}$$

avec $p \in \mathbb{R}_+^N$ vecteur des multiplicateurs, et $u = (u_i)_{i=1,\dots,N}$ et $(\theta_i)_{i=1,\dots,N}$ définis plus haut.
Regardons maintenant les Lagrangiens des sous-problèmes que l'on est amenés à résoudre dans la phase de décomposition des 3 algorithmes.

— Dans l'algorithme de décomposition par les prix, le Lagrangien des N sous-problèmes s'écrit :

$$\forall i \in \{1, \dots, N\}, \quad L(u_i) = J_i(u_i) + \langle p^{(k)}, \theta_i u_i \rangle$$

— Dans l'algorithme de décomposition par les quantités, le Lagrangien des N sous-problèmes s'écrit :

$$\forall i \in \{1, \dots, N\}, \quad L(u_i, \mu) = J_i(u_i) + \mu(\theta_i u_i - \omega_i^{(k)})$$

— Dans l'algorithme de décomposition par les prédictions, le Lagrangien du problème N s'écrit :

$$L(u_N, \mu) = J_N(u_N) + \mu(\theta_N u_N - v^{(k)})$$

et le Lagrangien des autres sous-problèmes s'écrit :

$$\forall i \in \{1, \dots, N-1\}, \quad L(u_i) = J_i(u_i) + \langle p^{(k)}, \theta_i u_i \rangle$$

Pour résoudre les sous-problèmes en pratique, nous pouvons utiliser des algorithmes de recherche de points selles comme les algorithmes d'Uzawa et d'Arrow-Hurwicz. Ces algorithme sont très facilement applicables à des problèmes quadratiques comme ceux que nous avons décrits en introduction mais ils utilisent des pas de gradient qu'il faut ajuster. Il est aussi parfois possible de trouver une solution explicite comme par exemple pour les sous-problèmes de la décomposition par les prix et par prédiction car ces dernier ne possèdent pas de contraintes (les résoudre revient donc à annuler le gradient de la fonction objectif). Enfin, nous pouvons également utiliser des méthodes de points intérieurs (décrites en introduction).

1.3 Vérification des conditions KKT

Dans cette partie, on va s'intéresser à la vérification des conditions de Karush–Kuhn–Tucker (KKT) au fur et à mesure des itérations de chaque méthode.
Ce problème ne présente pas de contraintes d'égalité. Aussi, il ne possède pas de multiplicateurs λ_i et les conditions KKT pour ce problème sont :

$$\begin{cases} \theta_i \, u_i \leq 0 & \forall i \in \{1, \dots, N\} \\ \mu_i \geq 0 & \forall i \in \{1, \dots, N\} \\ \mu_i \times \theta_i \, u_i = 0 & \forall i \in \{1, \dots, N\} \\ \nabla L(u, \mu) = 0 & \end{cases}.$$

En testant avec notre algorithme, et comme on pouvait s'y attendre, les conditions KKT sont satisfaites lorsque les algorithmes ont convergé. Toutefois, on observe également qu'à une itération quelconque avant convergence, ce n'est pas forcement le cas.

Par exemple, en testant à l'itération 10 pour les 3 méthodes, on s'aperçoit que pour la décomposition par prix les contraintes d'inégalité ne sont pas validées, pour la décomposition par les quantités c'est celle du produit des multiplicateurs et des contraintes d'inégalité ainsi que celle du gradient du Lagrangien qui ne sont pas validées. Pour la décomposition par prédiction, les contraintes d'inégalité et celle du produit entre multiplicateurs Mu et contraintes d'inégalité nul ne sont pas tenues.
De manière générale, les conditions ne sont pas toutes validées à chaque itération pour aucune des trois méthodes ; sauf lorsqu'elles ont convergé. Aussi, on pourrait se servir de ces conditions KKT comme critère d'arrêt pour nos algorithmes.

1.4 Comparaison des algorithmes

Ensuite, on va chercher à comparer les algorithmes correspondant aux trois méthodes, au travers des résultats obtenus, du temps d'exécution, du nombre d'itérations et de la complexité. Pour ce faire, on commence par tracer l'évolution de la solution retenue en fonction de l'itération considérée pour chaque méthode (pour un petit $N = 5$).

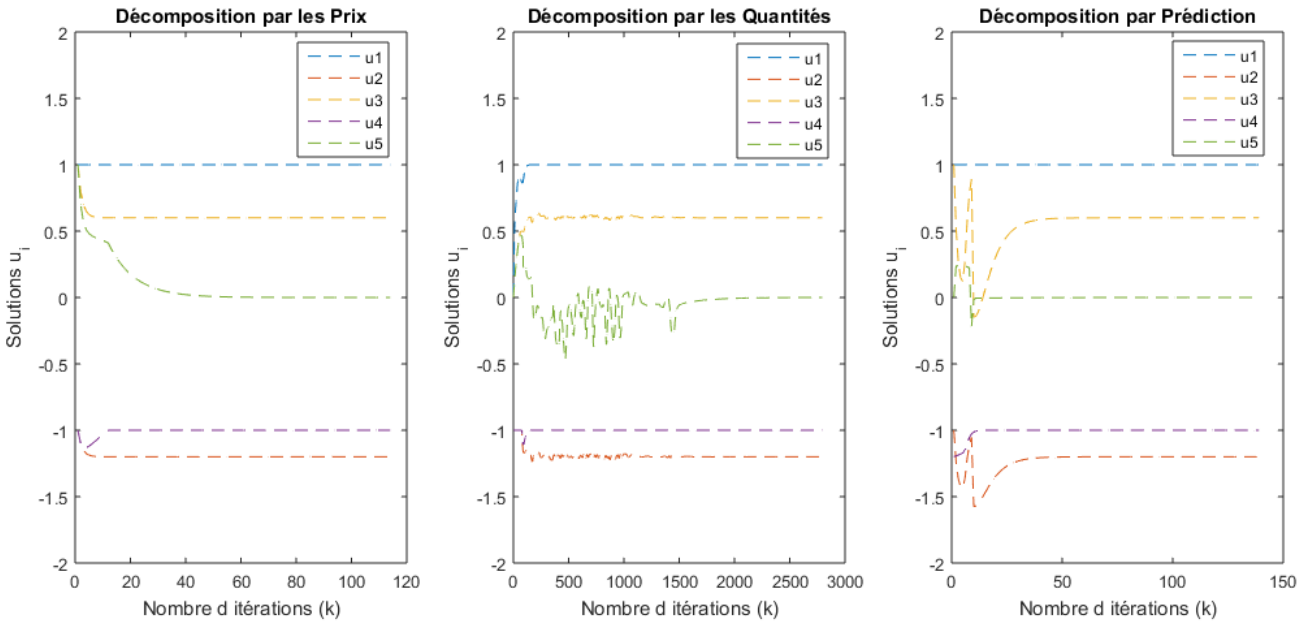

FIGURE 3 – Comparaison des solutions optimales renvoyées par les trois algorithmes

On constate que pour un faible nombre de variables, les méthodes par les prix et par prédiction semblent converger plus rapidement vers les solutions exactes que celle par allocation de ressources.

On cherche à confirmer cette intuition en comparant le temps d'exécution de chaque algorithme, ainsi que le nombre d'itérations réalisées. On récupère ces données en sortie d'algorithme pour plusieurs valeurs de N ; et on trace en fonction de N.

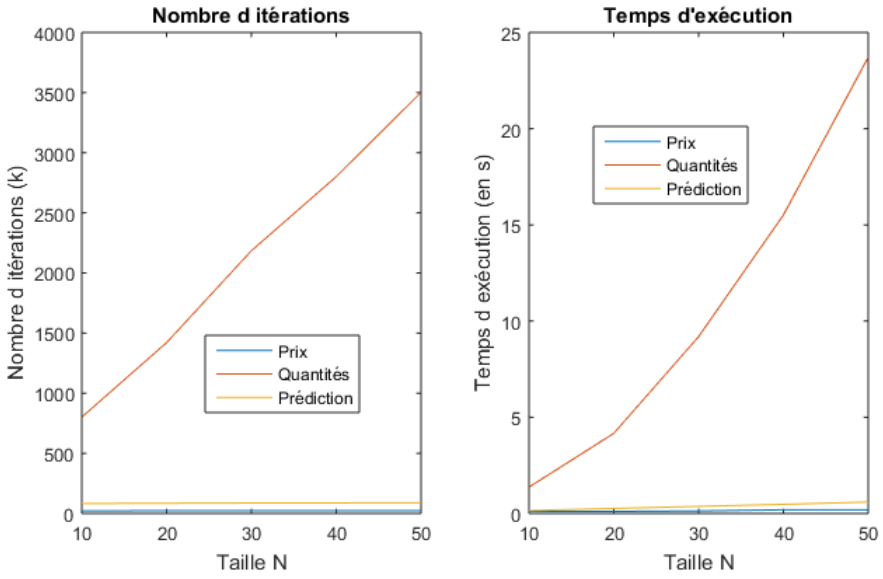

FIGURE 4 – Comparaison des temps d'exécution et nombre d'itérations des trois algorithmes

On constate un même phénomène sur ces deux graphiques : le nombre d'itérations et le temps d'exécution (logiquement liés) semblent varier très faiblement pour les décompositions par les prix et pas prédiction. On retrouve bien l'idée de réduire les calculs nécessaires en n'imposant pas de vérifier les contraintes à chaque itérations. Au contraire, l'algorithme de décomposition par quantités voit son temps d'exécution (et nombre d'itérations) augmenter fortement lorsque la taille du problème augmente. Cet algorithme semble donc moins efficace que les deux autres pour ce problème en particulier.

Enfin, pour estimer la complexité des algorithmes, on va chercher à estimer l'erreur entre la solution exacte et la solution approchée par nos algorithmes pour plusieurs valeurs de N. Pour ce faire, on calcule la solution exacte grâce à la fonction `fmincon` de MATLAB, qui est optimisée pour la résolution de problèmes de minimisation comme celui que nous traitons. Ensuite, on calcule la norme 2 de la différence entre solution approchée et celle calculée par `fmincon` : $erreur = \|u_{exact} - u_{calcule}\|_2$; et on la trace pour chaque valeur de N sélectionnée et pour chaque méthode.

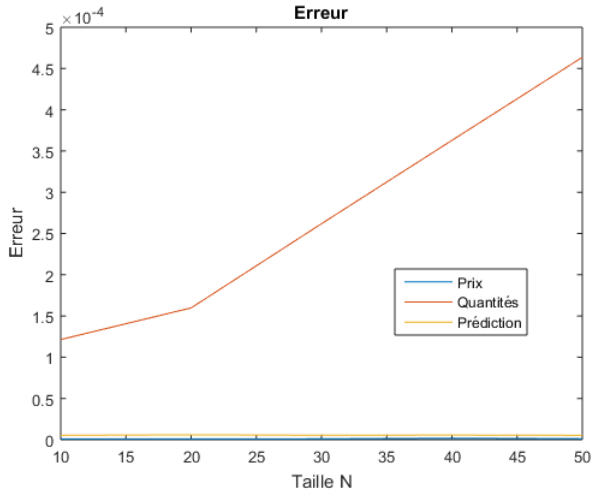

FIGURE 5 – Comparaison des erreurs pour les trois algorithmes

Là encore, on constate que la méthode de décomposition par les quantités semble avoir un résultat moins performant que les deux autres (qui ont une erreur constante et faible ici). L'erreur de cette méthode a davantage de variations lorsque le N augmente, ce qui montre qu'elle est moins précise.

1.5 Étude pour N grand (N = 200 et N = 251)

On se place dorénavant dans le cas où N est grand ($N = 200$ ou $N = 251$ selon les cas). On va ensuite chercher à étudier les variations de temps d'exécution et du nombre d'itérations pour chaque méthode dans le cas où la précision ϵ varie de 5×10^{-7} à 1×10^{-2}, et le pas ρ varie de 5×10^{-3} à 3.5×10^{-1}. Dans un deuxième temps, on va également faire varier les paramètres de relaxation β et γ entre 0 et 1 ; pour la méthode par prédiction.

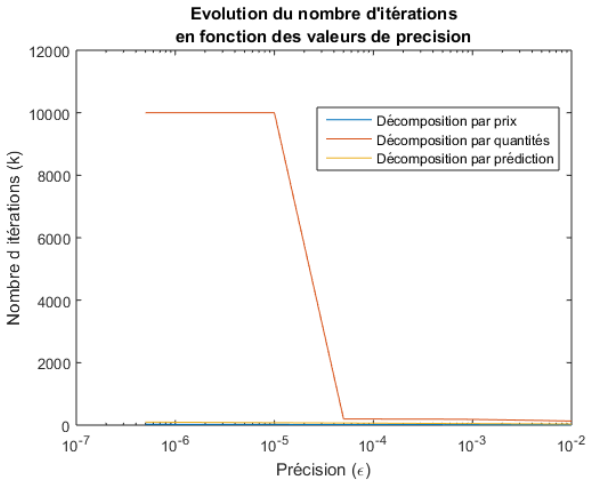

FIGURE 6 – Évolution du nombre d'itérations en fonction de la précision ϵ pour $N = 200$

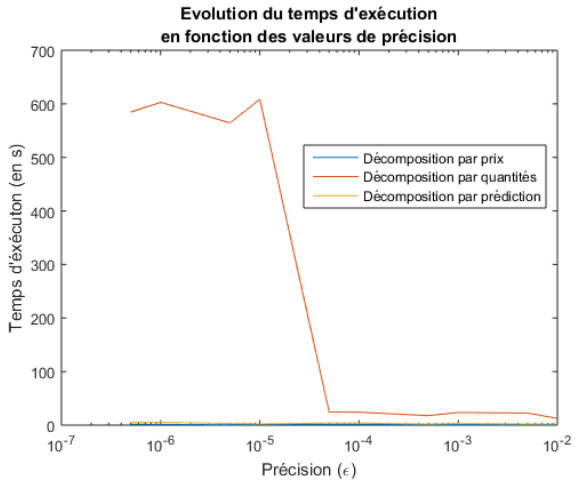

FIGURE 7 – Évolution du temps d'exécution en fonction de la précision ϵ pour $N = 200$

En observant l'évolution du nombre d'itérations et du temps d'exécution pour les trois méthodes, on s'aperçoit vite que le paramètre de précision ϵ a une grande influence pour l'algorithme de décomposition par les quantités. En effet, pour une précision plus fine que 10^{-4} on voit que le nombre d'itérations et le temps d'exécution explosent pour cet algorithme. Aussi, on sélectionnera une précision $\epsilon = 10^{-4}$ comme paramètre optimal de l'algorithme. On réalise ensuite la même analyse pour le pas ρ.

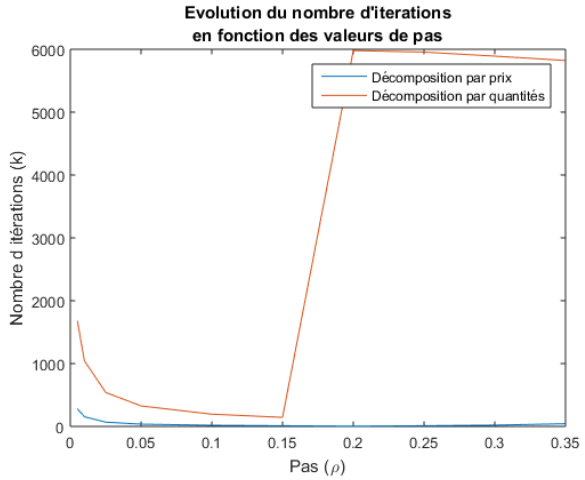

FIGURE 8 – Évolution du nombre d'itérations en fonction du pas ρ pour $N = 200$

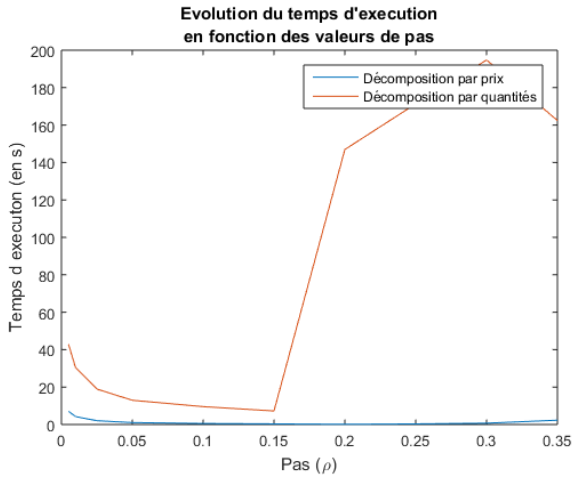

FIGURE 9 – Évolution du temps d'exécution en fonction du pas ρ pour $N = 200$

Là encore, on observe que pour un pas supérieur à 0.15, le nombre d'itérations et le temps d'exécution augmentent également beaucoup (même s'ils restent moindres que pour les variations de la précision). Comme précédemment, ces données restent plutôt stables pour la décomposition par les prix. On prendra donc une valeur $\rho = 0.1$ ou 0.15 pour nos résolutions. On s'intéresse enfin aux variations des paramètres de relaxation propres à la décomposition par prédiction.

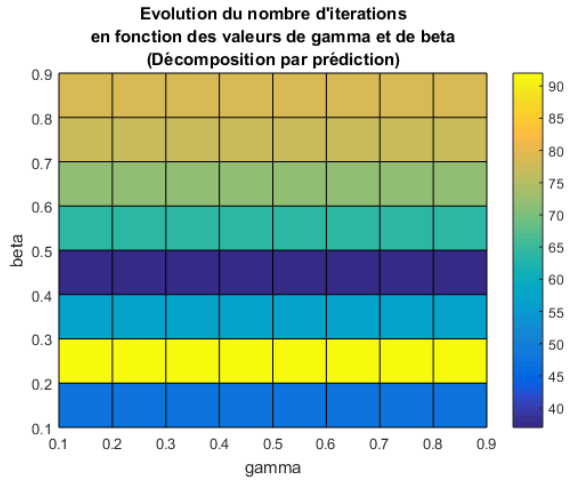

FIGURE 10 – Évolution du nombre d'itérations en fonction de β et γ pour $N = 200$

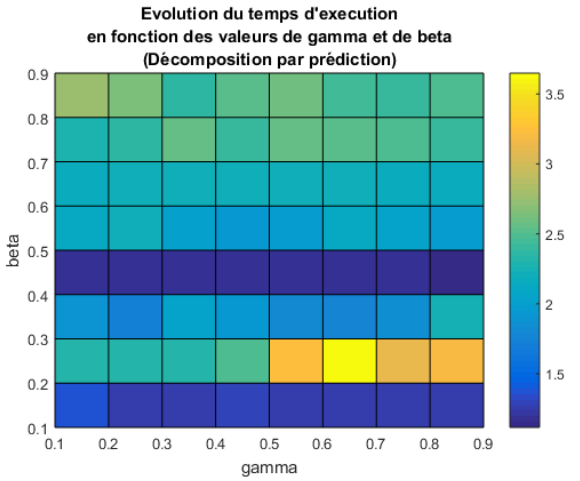

FIGURE 11 – Évolution du temps d'exécution en fonction de β et γ pour $N = 200$

Sur ces heatmaps, on peut visualiser graphiquement l'évolution du nombre d'itérations et du temps d'exécution en fonction simultanément des paramètres de relaxation, à l'aide de couleurs. On voit assez facilement que dans le cadre de notre problème et pour l'algorithme correspondant à la méthode de décomposition par prédiction, le paramètre γ ne semble pas beaucoup jouer, alors que β si, et il trouve une valeur optimale en 0.5.

On a pu conclure sur l'évolution des performances des trois algorithmes en fonction de divers paramètres, et trouver des valeurs optimales pour minimiser le temps d'exécution et le nombre d'itérations lorsque N devient grand. L'exécution pour $N = 251$ donne le même type de représentation mais avec un temps d'exécution beaucoup plus élevé et peut être visualisé en exécutant le code MATLAB.

1.6 Étude de l'ajout d'une surdiagonale

L'ajout d'un terme sur la surdiagonale tel que $A_{i,i+1} = -\alpha$, $\forall i \in 1...N-1$ $\alpha \in \mathbb{R}$ change la nature du problème considéré. En effet, en plus de modifier les propriétés de la matrice A selon la valeur de α, ceci rend l'objectif du problème couplant. Celui-ci n'est plus décomposable et nous ne pouvons plus traiter ce problème de manière directe par les méthodes de décomposition/coordination jusqu'ici étudiées : la décomposition par les prix, par les quantités ou par prédiction. Il faut alors résoudre ce problème par le principe du problème auxiliaire étudié en détail dans l'exercice 3 pour le modèle Markowitz.

Exercice 2 : Réseaux de distribution d'eau connectés

On s'intéresse au problème de l'optimisation de N sous-réseaux de distribution d'eau connectés par une usine de refoulement. Ce problème peut se mettre sous la forme suivante :

$$\begin{aligned} \min_{\{u_i=(u_i^1,u_i^2)\}_{1\leq i\leq N+1}} \quad & \sum_{i=1}^{N+1} J_i(u_i) \\ \text{subject to} \quad & \sum_{i=1}^{N} u_i - u_{N+1} = 0 \\ & u_i \in [0;a_i]\times[0;b_i-a_i] \qquad \forall i \in \{1,\dots,N\} \end{aligned} \tag{2}$$

On définit par ailleurs

— $J_{N+1}(u_{N+1}) = \frac{1}{2}\langle A_{N+1}u_{N+1}, u_{N+1}\rangle$

— $\forall i = 1,\dots,N$ $\begin{cases} J_i(u_i) = \min\limits_{v_i=(v_i^1,v_i^2)} & \frac{1}{2}\langle A_i v_i, v_i\rangle \\ \text{s. t} & -v_i^1 - u_i^1 + a_i \leq 0 \\ & v_i^1 + v_i^2 + u_i^1 + u_i^2 - b_i = 0 \end{cases}$

avec $(a_i)_i$ et $(b_i)_i$ tels que ($\forall i$, $b_i > a_i > 0$) et $A_i = \begin{pmatrix} \alpha_i & 0 \\ 0 & \beta_i \end{pmatrix}$ telles que ($\forall i$, $\alpha_i > 0$ et $\beta_i > 0$).

Écriture du problème (2) sous sa forme décomposable

Premièrement, afin de résoudre celui-ci, nous le réécrivons sous une forme standard et décomposable. Voici la nouvelle forme du problème 2 :

$$\begin{aligned} \min_{\{u_i\in U_i^{ad}\}_{i=1\dots N+1}} \quad & \sum_{i=1}^{N+1} J_i(u_i) \\ \text{s. t} \quad & \sum_{i=1}^{N+1} \theta_i(u_i) = 0 \end{aligned} \qquad \text{avec} \quad \begin{cases} \theta_i(u_i) = u_i \qquad \forall i = 1\dots N \\ \theta_{N+1}(u_{N+1}) = -u_{N+1} \\ U_i^{ad} = \left\{(u_1,u_2)\in\mathbb{R}^2 \mid u_1 \in [0;a_i],\ u_2 \in [0;b_i-a_i]\right\} \\ U_{N+1}^{ad} = \mathbb{R}^2 \end{cases}$$

Nous pouvons alors remarquer plusieurs choses :

— L'espace U est décomposable en produit cartésien de sous ensemble U_i. On observe la même chose pour les ensembles admissibles.

— La fonction objective et la contrainte couplante sont additives.

Par conséquent, nous allons pouvoir utiliser différentes méthodes de décomposition tout le long de l'exercice afin de résoudre le problème 2.

2.1 Décomposition par les prix

Dans cette première sous-partie, nous allons tout d'abord aborder la décomposition par les prix.

La décomposition par les prix est la plus simple des décompositions. Celle-ci dualise la contrainte couplante. Il ne reste ainsi plus qu'à résoudre le problème du point selle afin de déterminer la valeur optimale des u_i et du multiplicateur de Lagrange p associé à la contrainte couplante. Le multiplicateur p peut représenter le prix de perte ou de revente d'une sous ou surproduction associé à la contrainte couplante.

Pour cela, maintenant que nous avons décomposé notre problème principal 2 en plusieurs sous-problème i, nous devons minimiser chaque sous-problème $\mathbb{P}_i^k$ par rapport à u_i puis maximiser la fonction duale par un pas de gradient lors de la coordination. Voici un résumé des étapes de l'algorithme de la décomposition par les prix :

Décomposition Les sous-problèmes s'écrivent

$$\forall i = 1, \dots, N+1 \qquad \min_{u_i \in U_i^{ad}} J_i(u_i) + \langle p^{(k)}, \theta_i(u_i) \rangle \tag{$\mathbb{P}_i^k$}$$

Coordination On actualise les prix pour des contraintes d'égalité

$$p^{(k+1)} = p^{(k)} + \rho_k \left(\sum_{i=1}^{N} u_i^{(k+1)} - u_{N+1}^{(k+1)} \right)$$

Afin d'implémenter cette méthode sur MATLAB, nous allons d'abord expliciter les problèmes à résoudre lors de la décomposition.

Premièrement, nous pouvons réécrire le sous-problème pour $i \in \{1 \dots N\}$ car il s'agit d'un problème min min :

$$\forall i = 1, \dots, N \quad \begin{cases} \min\limits_{u_i \in \mathbb{R}^2, v_i \in \mathbb{R}^2} & \frac{1}{2}\langle A_i v_i, v_i \rangle + \langle p^{(k)}, u_i \rangle \\ \text{s.t} & 0 \leq u_i^1 \leq a_i \\ & 0 \leq u_i^2 \leq b_i - a_i \\ & -v_i^1 - u_i^1 + a_i \leq 0 \\ & v_i^1 + v_i^2 + u_i^1 + u_i^2 - b_i = 0 \end{cases}$$

Pour résoudre ce problème, nous allons utiliser plusieurs méthodes :

- l'algorithme d'Arrow-Hurwicz afin de résoudre le problème par rapport à u_i et v_i. Malheureusement, ici nous ne pouvons pas utiliser l'algorithme d'Uzawa quadratique que nous avons développé. En effet, la matrice A utilisée dans celui-ci n'est pas inversible dans le sous-problème $\mathbb{P}_i^k$.
- la méthode des points intérieurs

Deuxièmement, pour $i = N+1$ on trouve une solution directe en annulant le gradient comme u_{N+1} est libre dans $\mathbb{R}^2$:

$$\begin{aligned} & \min_{u_{N+1}\in\mathbb{R}^2} \quad J_{N+1}(u_{N+1}) - \langle p^{(k)}, u_{N+1}\rangle \\ \iff & \min_{u_{N+1}\in\mathbb{R}^2} \quad \frac{1}{2}\left(\alpha_{N+1}(u^1_{N+1})^2 + \beta_{N+1}(u^2_{N+1})^2\right) - p_1^{(k)}u^1_{N+1} - p_2^{(k)}u^2_{N+1} \\ \implies & \quad \begin{cases} (u^1_{N+1})^{(k+1)} = \dfrac{p_1^{(k)}}{\alpha_{N+1}} \\ (u^2_{N+1})^{(k+1)} = \dfrac{p_2^{(k)}}{\beta_{N+1}} \end{cases} \end{aligned}$$

Nous avons ainsi explicité chaque sous problème $\mathbb{P}_i^k$ et savons combiner leurs solutions par la formule de coordination. Par conséquent, nous pouvons implémenter l'algorithme de décomposition par les prix 3 décrit au premier exercice appliqué au problème 2 sur MATLAB.

Test exemple jouet Afin de tester la validité de notre algorithme, nous le testons sur un exemple jouet sur lequel nous testerons chaque méthode afin de voir leurs inconvénients et avantages.

Nous prenons N=1 et les paramètres suivants :

- $a_1 = 1$ et $b_1 = 2$
- $\alpha_1 = 4$ et $\beta_1 = 5$
- $\alpha_2 = 1$ et $\beta_1 = 2$

Nous obtenons ainsi le problème suivant :

$$\begin{cases} \min\limits_{u_1\in[0,1]^2, u_{N+1}, v_1\in\mathbb{R}^2} & \frac{1}{2}(4{v_1^1}^2 + 5{v_1^2}^2) + \frac{1}{2}({u_2^1}^2 + 2{u_2^2}^2) \\ \text{s.t} & u_1^1 - u_2^1 = 0 \\ & u_1^2 - u_2^2 = 0 \\ & -v_1^1 - u_1^1 + 1 \leq 0 \\ & v_1^1 + v_1^2 + u_1^1 + u_1^2 - 2 = 0 \end{cases}$$

La solution optimale de ce dernier est :

- $J^*_{opt} = \frac{39}{38} \approx 1.0263$
- $u_1^* = u_2^* = \begin{pmatrix} 1 & \frac{10}{19} \end{pmatrix} \approx \begin{pmatrix} 1 & 0.526 \end{pmatrix}$
- $v_1^* = \begin{pmatrix} \frac{4}{19} & \frac{5}{19} \end{pmatrix} \approx \begin{pmatrix} 0.211 & 0.263 \end{pmatrix}$

Nous tentons alors de résoudre le problème par l'algorithme de décomposition par prix avec les 2 méthode de résolutions (Arrow-Hurwicz et points intérieurs), nous obtenons les résultats suivants :

$$u_1^{prix} = u_2^{prix} = \begin{pmatrix} 1 & 0.526 \end{pmatrix} \texttt{ et } v_1^{prix} = \begin{pmatrix} 0.211 & 0.263 \end{pmatrix}$$

Nos résultats sont donc très satisfaisants l'algorithme a convergé vers la bonne solution. De plus, l'exécution de la décomposition par les prix a nécessité seulement 7 itérations et 0.017995 secondes. Voici quelques graphes résumant la résolution du problème jouet avec la décomposition par prix :

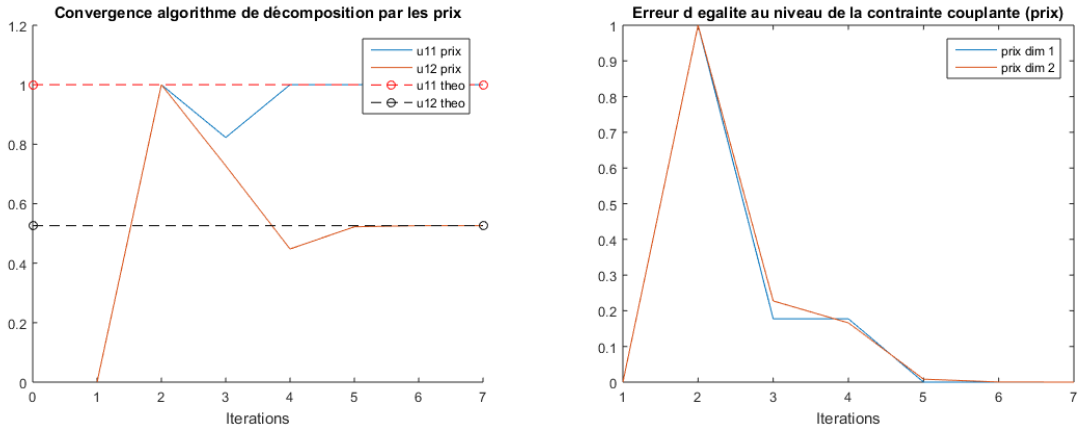

FIGURE 12 – Analyse résolution exemple jouet par la décomposition par les prix

Premièrement, sur le graphique de gauche, nous avons représenté les valeurs de u_1 en fonction des itérations de l'algorithme. Nous observons que très vite les valeurs se rapprochent des valeurs théoriques. L'algorithme de décomposition par les prix converge donc plutôt vite. Nous pourrons comparer la vitesse par la suite avec les autres.

Deuxièmement, sur le graphique de droite, nous avons représenté les différences $u_1^1 - u_2^1$ et $u_1^2 - u_2^2$ qui représentent la contrainte couplante et qui doivent donc être nulle afin de respecter celle-ci et donc avoir une solution réalisable. Nous observons alors que l'algorithme de décomposition par les prix ne respecte pas la contrainte couplante à chaque itération. Afin d'obtenir une solution réalisable, nous devons donc attendre la convergence de l'algorithme.

Nous allons maintenant dans la prochaine sous-section utiliser un autre algorithme de décomposition.

2.2 Décomposition par les quantités

Dans cette deuxième sous-section, nous allons nous intéresser à la décomposition par quantités ou allocation des ressources.

Au lieu d'agir sur un prix, la décomposition par quantité oblige chaque sous-problème $\mathbb{Q}_i^k$ à produire une certaine quantité donnée en faisant sorte qu'à chaque itération, nous respectons la contrainte couplante.

Pour cela, nous devons résoudre chaque sous-problème $\mathbb{Q}_i^k$ telle que u_i soit égale à une quantité w_i puis nous cherchons alors à minimiser la valeur de la fonction objective en effectuant un pas de gradient projeté des quantités w sur l'espace respectant la contrainte couplante c'est à dire telle que $\sum_{i=1}^{N} w_i - w_{N+1} = 0$. Voici un résumé des étapes de l'algorithme de la décomposition par les quantités :

Décomposition Les sous-problèmes s'écrivent

$$\forall i = 1, \dots, N+1 \quad \begin{cases} \min\limits_{u_i \in U_i^{ad}} & J_i(u_i) \\ \text{s.t} & \theta_i(u_i) = \omega_i^k \end{cases} \tag{$\mathbb{Q}_i^k$}$$

À partir desquels on récupère les solutions $u_i^{(k+1)}$ et les multiplicateurs $p_i^{(k+1)}$.

Coordination On actualise les quantités pour des contraintes d'égalité

$$\omega_i^{(k+1)} = \omega_i^{(k)} + \varepsilon_k \left(p_i^{(k+1)} - \frac{1}{N+1} \sum_{j=1}^{N+1} p_j^{(k+1)} \right)$$

Afin d'implémenter ce dernier, nous allons expliciter la résolution des sous-problèmes.

Premièrement, le sous-problème pour $i \in \{1 \dots N\}$ est le suivant :

$$\forall i = 1, \dots, N \quad \begin{cases} \min\limits_{u_i \in \mathbb{R}^2, v_i \in \mathbb{R}^2} & \frac{1}{2}\langle A_i v_i, v_i \rangle + \langle p^{(k)}, u_i \rangle \\ \text{s.t} & u_i^1 = \omega_{i,1}^k \\ & u_i^2 = \omega_{i,2}^k \\ & 0 \leq u_i^1 \leq a_i \\ & 0 \leq u_i^2 \leq b_i - a_i \\ & -v_i^1 - u_i^1 + a_i \leq 0 \\ & v_i^1 + v_i^2 + u_i^1 + u_i^2 - b_i = 0 \end{cases}$$

Nous pouvons alors remarquer que la valeur des u_i est fixé, nous pouvons donc seulement résoudre un problème de minimisation sur les v_i.

En revanche, même si u_i est fixé il doit aussi appartenir à U_i^{ad}. Par conséquent, si w_i^k n'appartient pas à U_i^{ad} alors le sous-problème $\mathbb{Q}_i^k$ n'admet pas de solution et donc l'algorithme ne convergera pas.

Dans notre cas, dans un premier temps, nous décidons de relâcher les contraintes sur l'ensemble admissible afin de tester si l'algorithme par quantité convergera quand même vers la bonne solution. Cela arrivera seulement si la solution optimale du problème appartient à l'ensemble admissible des U^{ad}.

Néanmoins, nous devons tout d'abord récupérer les valeurs des $p_i^{(k+1)}$ associées aux contraintes d'allocation dans chaque sous problème $\mathbb{Q}_i^k$ pour la coordination en enlevant les contraintes sur l'ensemble admissible. Nous nommons respectivement p_i^1 , p_i^2 les contraintes associées aux contraintes d'allocations et μ et λ les contraintes associées à l'inégalité $-v_i^1 - u_i^1 + a_i \leq 0$ et l'égalité $v_i^1 + v_i^2 + u_i^1 + u_i^2 - b_i = 0$.

Par les conditions de Karush-Kuhn-Tucker, nous trouvons que

$$p_i^* = \begin{pmatrix} \mu^* - \lambda^*, & -\lambda^* \end{pmatrix}$$

Suite à la suppression des contraintes sur l'ensemble admissible de U_i, nous obtenons directement la valeur de u_i et nous pouvons réécrire chaque sous-problème $\mathbb{Q}_i^k$ de la manière suivante comme un problème de minimisation sur v_i seulement :

$$\forall i = 1, \ldots, N \quad \begin{cases} \min\limits_{v_i \in \mathbb{R}^2} & J_i(v_i) \\ \text{s.t} & -v_i^1 - \omega_{i,1}^k + a_i \leq 0 \\ & v_i^1 + v_i^2 + \omega_{i,1}^k + \omega_{i,2}^k - b_i = 0 \end{cases}$$

Ce sous-problème i devient alors simple. Il y a seulement une contrainte d'inégalité et une contrainte d'égalité. Il est donc raisonnable d'utiliser les conditions de Karush-Kuhn-Tucker afin de déterminer des solutions explicites pour les valeurs optimales.

Ainsi, voici les conditions KKT associées à chaque sous-problème i :

$$\forall i = 1, \ldots, N \quad \begin{cases} \alpha_i v_i^{1^*} - \mu^* + \lambda^* = 0 \\ \beta_i v_i^{2^*} + \lambda^* = 0 \\ v_i^{1^*} + v_i^{2^*} + \omega_{i,1}^k + \omega_{i,2}^k - b_i = 0 \\ \mu^*(-v_i^{1^*} - \omega_{i,1}^k + a_i) = 0 \\ \mu^* \geq 0 \ \texttt{et} \ -v_i^{1^*} - \omega_{i,1}^k + a_i \leq 0 \end{cases}$$

Après résolution, nous trouvons les solutions suivantes :

Cas n°1 :

$$\frac{\omega_{i,1}^k + \omega_{i,2}^k - b_i}{\alpha_i + \beta_i}\beta_i - \omega_{i,1}^k + a_i \leq 0 \Rightarrow \begin{cases} v_i^{1^*} = \dfrac{(b_i - \omega_{i,1}^k - \omega_{i,2}^k)\beta_i}{\alpha_i + \beta_i} \\ v_i^{2^*} = \dfrac{(b_i - \omega_{i,1}^k - \omega_{i,2}^k)\alpha_i}{\alpha_i + \beta_i} \\ \lambda^* = \dfrac{(\omega_{i,1}^k + \omega_{i,2}^k - b_i)\alpha_i\beta_i}{\alpha_i + \beta_i} \\ \mu^* = 0 \end{cases}$$

Cas n°2 :

$$\frac{\omega_{i,1}^k + \omega_{i,2}^k - b_i}{\alpha_i + \beta_i}\beta_i - \omega_{i,1}^k + a_i > 0 \Rightarrow \begin{cases} v_i^{1^*} = a_i - \omega_{i,1}^k \\ v_i^{2^*} = b_i - a_i - \omega_{i,2}^k \\ \lambda^* = (a_i + \omega_{i,2}^k - b_i)\beta_i \\ \mu^* = \alpha_i(a_i - \omega_{i,1}^k) + (a_i + \omega_{i,2}^k - b_i)\beta_i \end{cases}$$

Pour résoudre ce problème, nous allons fixer la valeur de u_i à la valeur des $w_i^{(k)}$ puis utiliser plusieurs méthodes pour résoudre le problème de minimisation sur v_i :

— l'algorithme d'Uzawa
— la méthode des points intérieurs
— résolution par les formules explicites

Deuxièmement, pour $i = N + 1$ on obtient le problème suivant :

$$\begin{aligned} \min_{u_{N+1} \in \mathbb{R}^2} \quad & J_{N+1}(u_{N+1}) = \frac{1}{2}(\alpha_{N+1} u_{N+1}^{1^2} + \beta_{N+1} u_{N+1}^{2^2}) \\ \text{s.t} \quad & -u_{N+1} = \omega_{N+1}^{(k)} \end{aligned}$$

Grâce aux conditions de Karush-Kuhn-Tucker, nous pouvons résoudre le sous-problème $N+1$ avec les formules explicites suivantes :

— $u_{N+1}^{(k+1)} = -\omega_{N+1}^{(k)}$
— $p_{1,N+1}^{(k+1)} = -\alpha_{N+1}\omega_{1,N+1}^{(k)} = \alpha_{N+1} u_{1,N+1}^{(k+1)}$
— $p_{2,N+1}^{(k+1)} = -\beta_{N+1}\omega_{2,N+1}^{(k)} = \beta_{N+1} u_{2,N+1}^{(k+1)}$

Nous avons ainsi explicité chaque sous problème $\mathbb{Q}_i^k$ et nous avons une formule de coordination. Par conséquent, nous pouvons implémenter l'algorithme de décomposition par les quantités décrit au premier exercice sur MATLAB au problème 2.

Exemple Jouet Nous tentons alors de résoudre l'exemple jouet par l'algorithme de décomposition par quantités avec les 3 méthodes de résolution (Uzawa, points intérieurs et formule explicite), nous obtenons les résultats suivants :

$$u_1^{quantites} = u_2^{quantites} = \begin{pmatrix} 1.0257 & 0.5128 \end{pmatrix} \text{ et } v_1^{quantites} = \begin{pmatrix} 0.2564 & 0.2051 \end{pmatrix}$$

Nos résultats ne sont pas bons. L'algorithme a convergé en 24 itérations et 0.10 secondes (Uzawa) mais pas vers la bonne solution. La valeur de la fonction objective est égale a 1.0256 qui est donc inférieure à la valeur optimale du problème jouet mais malheureusement la solution renvoyée par l'algorithme n'appartient pas à l'ensemble admissible des U^{ad}. Cela s'explique par le fait que nous avons supprimé les contraintes sur l'ensemble admissible de U dans chaque sous problème $\mathbb{Q}_i^k$.

Pour pallier à ce problème, nous allons créer un deuxième algorithme de décomposition par les quantités tenant compte de l'ensemble admissible de U^{ad}.

Pour cela, nous allons seulement réaliser quelques petits changements. Tout d'abord, chaque sous problème $\mathbb{Q}_i^k$ reste le même, nous reportons seulement les contraintes admissibles de u sur les ω pour les N premiers sous-problèmes. Voici la nouvelle formulation des sous-problèmes :

$$\forall i = 1, \dots, N \quad \begin{cases} \min\limits_{u_i \in \mathbb{R}^2, v_i \in \mathbb{R}^2} & \frac{1}{2}\langle A_i v_i, v_i \rangle + \langle p^{(k)}, u_i \rangle \\ \text{s.t} & u_i^1 = \omega_{i,1}^k \\ & u_i^2 = \omega_{i,2}^k \\ & 0 \leq \omega_{i,1}^k \leq a_i \\ & 0 \leq \omega_{i,2}^k \leq b_i - a_i \\ & -v_i^1 - u_i^1 + a_i \leq 0 \\ & v_i^1 + v_i^2 + u_i^1 + u_i^2 - b_i = 0 \end{cases}$$

Le problème devient alors simple et est simplement un problème de minimisation sur v qui est le même qu'à la première partie. Nous pouvons donc réutiliser les différents algorithmes que nous avions cités pour résoudre le problème de minimisation sur v. Néanmoins, nous devons trouver un moyen pour que les ω respecte bien les contraintes admissibles de u lors de la coordination et que les ω respecte la contrainte couplante.

Pour simplifier le pas de gradient projeté et ne pas avoir à projeter sur une intersection, nous décidons de fixer la valeur $w_{N+1}^{(k)}$ à l'opposé de la somme des autres ω. Cela oblige les ω à respecter la contrainte couplante à chaque itération. Ainsi, voici la décomposition associée au N+1-ème sous-problème :

$$\begin{aligned} \min_{u_{N+1} \in \mathbb{R}^2} \quad & J_{N+1}(u_{N+1}) = \frac{1}{2}(\alpha_{N+1} u_{N+1}^{1^2} + \beta_{N+1} u_{N+1}^{2^2}) \\ \text{s.t} \quad & -u_{N+1} = \omega_{N+1}^{(k)} = -\sum_{i=1}^{N} \omega_i \end{aligned}$$

Les solutions restent les mêmes que précédemment :
- $u_{N+1}^{(k+1)} = -\omega_{N+1}^{(k)} = \sum_{i=1}^{N} \omega_i$
- $p_{1,N+1}^{(k+1)} = -\alpha_{N+1}\omega_{1,N+1}^{(k)} = \alpha_{N+1} u_{1,N+1}^{(k+1)}$
- $p_{2,N+1}^{(k+1)} = -\beta_{N+1}\omega_{2,N+1}^{(k)} = \beta_{N+1} u_{2,N+1}^{(k+1)}$

Nous devons terminer par modifier l'étape de coordination des ω. Cette fois-ci, nous devons effectuer un pas de gradient projeté sur l'ensemble admissible de U. Maintenant, pour chaque $i \in 1, .., N$, chaque ω_i appartient au ième sous problème et N+1-ème sous problème. Il faut donc calculer le gradient par rapport à ω_i dans les 2 sous-problèmes. Pour le ième problème, le gradient reste le même et est égal à $-p_i$. Pour le N+1-ème sous-problème, nous trouvons par KKT que le gradient par rapport à ω_i est égale à p_{N+1}.

Nous pouvons alors réécrire l'étape de coordination de la manière suivante en projetant sur l'ensemble admissible :

$$\omega_i^{(k+1)} = \max\left[\min\left[\omega_i^{(k)} + \varepsilon_k\left(p_i^{(k+1)} - p_{N+1}^{(k+1)}\right), \begin{pmatrix} a_i \\ b_i - a_i \end{pmatrix}\right], \begin{pmatrix} 0 \\ 0 \end{pmatrix}\right], \quad \forall i \in 1, .., N$$

Nous avons ainsi explicité chaque sous problème $\mathbb{Q}_i^k$ et nous avons une formule de coordination. Par conséquent, nous pouvons implémenter sur MATLAB un deuxième algorithme de décomposition par les quantités (4) que nous nommons `quantites2`.

Exemple Jouet Nous tentons alors de résoudre l'exemple jouet par l'algorithme de décomposition par quantités avec les 3 méthodes de résolution (Uzawa, points intérieurs et formule explicite), nous obtenons les résultats suivants :

$$u_1^{quantites2} = u_2^{quantites2} = \begin{pmatrix} 1 & 0.526 \end{pmatrix} \quad \text{et} \quad v_1^{quantites2} = \begin{pmatrix} 0.211 & 0.263 \end{pmatrix}$$

Nos résultats sont cette fois-ci bons. L'algorithme a convergé en 18 itérations et 0.05 secondes (Uzawa) et vers la bonne solution.

Afin de comparer les 2 versions et de résumer la résolution du problème jouet avec la décomposition par quantités, voici quelques graphes :

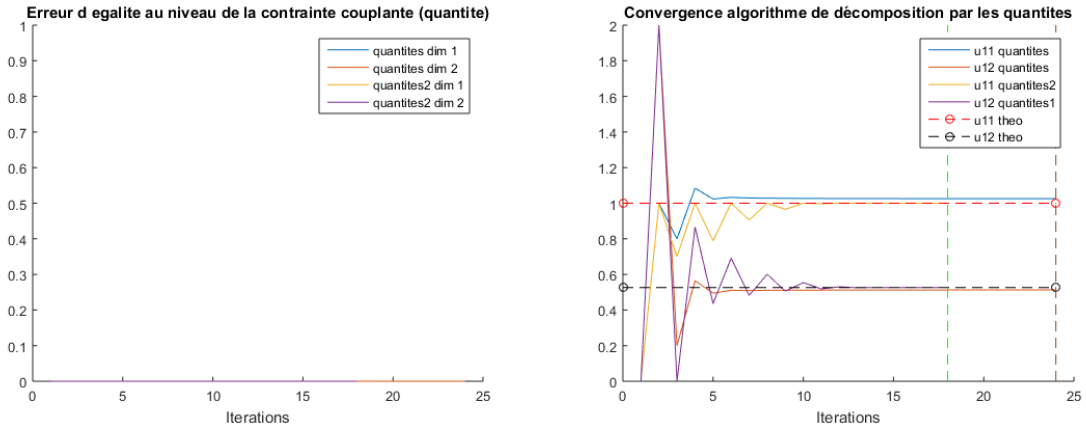

FIGURE 13 – Analyse résolution exemple jouet par la décomposition par les quantites

Premièrement, sur le graphique de gauche, nous avons représenté les valeurs de u_1 en fonction des itérations de l'algorithme pour les 2 algorithmes de décomposition par quantités. Nous observons que les algorithmes convergent assez vite vers des valeurs (`quantites2` converge un peu plus vite). En revanche, le premier algorithme de decomposition par quantité ne converge pas vers la bonne solution alors que le deuxième converge vers la bonne solution. Par conséquent, par la suite, nous utiliserons seulement le 2ème algorithme de décomposition par les quantités. Nous pourrons comparer la vitesse avec les autres par la suite mais l'avantage de la décomposition par

les quantités est que nous pouvons utiliser une résolution des sous-problèmes avec des formules explicites ce qui peut diminuer la durée d'une itération.

Deuxièmement, sur le graphique de droite, nous avons représenté les différences $u_1^1 - u_2^1$ et $u_1^2 - u_2^2$ qui représentent la contrainte couplante et qui doivent donc être nulles afin de respecter celle-ci et donc avoir une solution réalisable. Nous observons alors que l'algorithme de décomposition par les quantités respecte la contrainte couplante à chaque itération. C'est donc un des avantages par rapport à la décomposition par les prix, nous pouvons nous arrêter à n'importe quelle itération et nous aurons une solution réalisable.

Nous allons maintenant dans la prochaine sous-section utiliser un dernier autre algorithme de décomposition.

2.3 Décomposition par prédiction

Dans cette troisième sous-section, nous allons nous intéresser à la décomposition par prédiction.

L'objectif de la méthode de décomposition par les prédictions est de pallier les inconvénients des décompositions par les quantités et par les prix. Cette méthode fera en sorte de partitionner l'ensemble des contraintes du problème 2 entre les différents sous-problèmes.

Nous devons donc choisir un certain sous-problème auquel nous affecterons la contrainte couplante. Dans notre cas, nous choisissons le sous-problème N+1. Les autres sous-problèmes, eux, ne verront pas la contrainte couplante mais nous leur ajoutons un terme d'incitation à produire p comme dans la décomposition par les prix.

Voici un résumé des étapes de l'algorithme de la décomposition par prédiction :

Résolution du sous-problème $N+1$

$$\begin{array}{ll} \min\limits_{u_{N+1}\in\mathbb{R}^2} & J_{N+1}(u_{N+1}) \\ \text{s.t} & -u_{N+1} = v^{(k)} \end{array} \qquad (\mathbb{R}^k_{N+1})$$

On récupère les multiplicateurs de Lagrange $\lambda^{(k+1)} = (\lambda_1^{(k+1)}, \lambda_2^{(k+1)})^\top$ associés à la contrainte d'allocation et $u_{N+1}^{(k+1)}$.

Résolution des sous-problèmes restants

$$\forall i = 1, \dots, N \quad \begin{cases} \min\limits_{u_i\in U_i^{ad},\, v_i\in\mathbb{R}^2} & J_i(v_i) + \langle p^{(k)}, u_i\rangle \\ \text{s.t} & -v_i^1 - u_i^1 + a_i \le 0 \\ & v_i^1 + v_i^2 + u_i^1 + u_i^2 - b_i = 0 \end{cases} \qquad (\mathbb{R}^k_i)$$

On récupère $u_1^{(k+1)}$,...,$u_N^{(k+1)}$.

Calcul des prix On réalise une relaxation pour l'actualisation des prix

$$p^{(k+1)} = (1-\beta)p^{(k)} + \beta\lambda^{(k+1)}$$

Calcul de l'allocation On réalise une relaxation pour l'actualisation de l'allocation

$$v^{(k+1)} = (1-\gamma)v^{(k)} - \gamma\sum_{i=1}^{N}\theta_i u_i^{(k+1)}$$

Nous remarquons qu'il y a plusieurs étapes, il y en a tout 4 : décomposition du N+1-ème sous problème, décompositions des sous-problèmes restants, calcul des prix et calcul de l'allocation. Nous allons voir que nous pouvons donc développer différents algorithmes de prédiction en fonction de l'ordre que nous choisissons pour les étapes mais tout d'abord nous allons expliciter la résolution des sous-problèmes.

Premièrement, pour la résolution du N+1-ème sous problème, on obtient le problème suivant :

$$\begin{aligned} \min_{u_{N+1}\in\mathbb{R}^2} \quad & J_{N+1}(u_{N+1}) = \frac{1}{2}(\alpha_{N+1}u_{N+1}^{1^2} + \beta_{N+1}u_{N+1}^{2^2}) \\ \text{s.t} \quad & -u_{N+1} = v^{(k)} \end{aligned}$$

C'est le même que celui de la décomposition par quantités. Par conséquent, grâce aux conditions de Karush-Kuhn-Tucker, nous pouvons résoudre le sous-problème $N+1$ avec les formules explicites suivantes :

- $u_{N+1}^{(k+1)} = -v^{(k)}$
- $p_{1,N+1}^{(k+1)} = -\alpha_{N+1}v_1^{(\mathrm{k})} = \alpha_{N+1}u_{1,N+1}^{(k+1)}$
- $p_{2,N+1}^{(k+1)} = -\beta_{N+1}v_2^{(\mathrm{k})} = \beta_{N+1}u_{2,N+1}^{(k+1)}$

Deuxièmement, pour la résolution des N sous-problèmes restants, nous avons les problèmes suivants :

$$\forall i = 1,\dots,N \quad \begin{cases} \min\limits_{u_i\in\mathbb{R}^2, v_i\in\mathbb{R}^2} & \frac{1}{2}\langle A_i v_i, v_i\rangle + \langle p^{(k)}, u_i\rangle \\ \text{s.t} & 0 \le u_i^1 \le a_i \\ & 0 \le u_i^2 \le b_i - a_i \\ & -v_i^1 - u_i^1 + a_i \le 0 \\ & v_i^1 + v_i^2 + u_i^1 + u_i^2 - b_i = 0 \end{cases}$$

Ces sont les mêmes problèmes que la décompotion par les prix. Par conséquent, pour résoudre ce problème, nous allons utiliser plusieurs méthodes :

— l'algorithme d'Arrow-Hurwicz afin de résoudre le problème par rapport à u_i et v_i. Malheureusement, ici nous ne pouvons pas utiliser l'algorithme d'Uzawa quadratique que nous avons développé. En effet, la matrice A utilisée dans celui-ci n'est pas inversible dans le sous-problème $\mathbb{R}_i^k$...
— la méthode des points intérieurs

Nous avons donc explicité la résolutions des N+1-sous problèmes, nous allons maintenant nous intéresser à l'ordre des étapes. Grâce à l'ordre des étapes, nous allons pouvoir créer 3 algorithmes différents :

Décomposition par prédiction parallèle Le premier algorithme utilise une programmation parallèle de la résolution. Nous réalisons tout d'abord la résolution du N+1-ème problème ainsi que des N autres sous-problèmes restants puis nous effectuons les coordinations sur les allocations et sur les prix. Nous nommons cet algorithme décomposition par prédiction `par`.

Décomposition par prédiction séquentielle 1 et 2 Les deux algorithmes suivants vont cette fois-ci utiliser une programmation séquentielle. Le premier commencera par résoudre les N sous problèmes, effectuera la coordination sur l'allocation, résoudra le N+1ème sous problème et enfin effectuera la coordination sur les prix. Nous le nommons décomposition par prédiction `seq`.
La deuxième commencera tout d'abord par résoudre le N+1ème sous problème. Il effectuera ensuite une coordination sur les prix. Puis, il résoudra les N autres sous problèmes et enfin il effectuera la coordination sur l'allocation. Nous le nommons décomposition par prédiction `seq2`.

Algorithme Nous implémentons ces trois versions de l'algorithme de décomposition par prédiction sur MATLAB.

Exemple Jouet Nous tentons alors de résoudre l'exemple jouet par l'algorithme de prédiction par quantités avec les 2 méthodes de résolution (Arrow-Hurwicz et points intérieurs), nous obtenons les mêmes résultats pour les 3 algorithmes :

$$u_1^{prediction} = u_2^{prediction} = \begin{pmatrix} 1 & 0.526 \end{pmatrix} \texttt{ et } v_1^{prediction} = \begin{pmatrix} 0.211 & 0.263 \end{pmatrix}$$

Nos résultats sont bons, les 3 algorithmes ont convergé vers la bonne solution. En revanche, les nombres d'itérations sont différents :
— Décomposition par prédiction `par` : 33 itérations et 0.13 secondes (Arrow-Hurwicz)
— Décomposition par prédiction `seq` : 8 itérations et 0.06 secondes (Arrow-Hurwicz)
— Décomposition par prédiction `seq2` : 8 itérations et 0.06 secondes (Arrow-Hurwicz)

Nous remarquons alors que les algorithmes séquentielles se ressemblent beaucoup et vont plus vite que l'algorithme parallèle. Nous étudierons plus en détails ces différences dans une prochaine partie.

Afin de comparer les 3 versions et de résumer la résolution du problème jouet avec la décomposition par prédiction, voici quelques graphes :

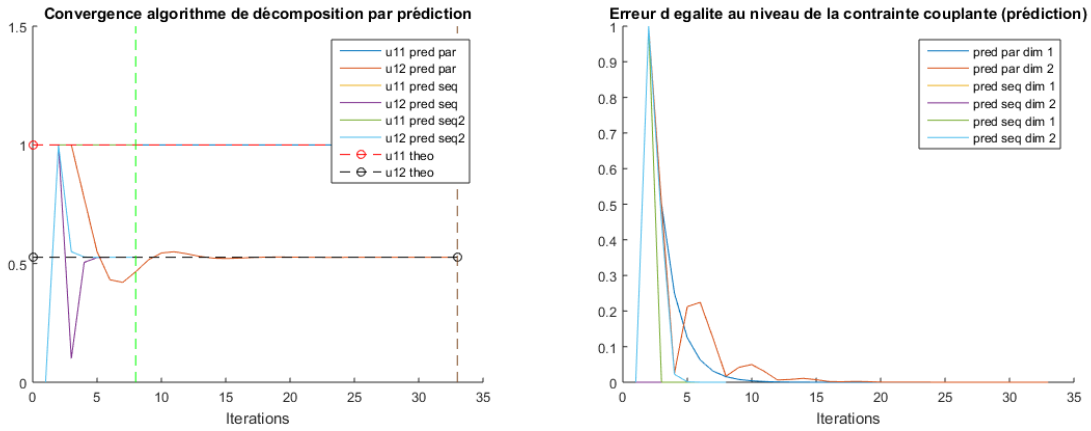

FIGURE 14 – Analyse résolution exemple jouet par la décomposition par prédiction

Premièrement, sur le graphique de gauche, nous avons représenté les valeurs de u_1 en fonction des itérations de l'algorithme. Nous observons que les algorithmes séquentiels convergent plus vite vers la solution. Nous verrons dans la prochaine partie que la convergence est influencée par les paramètres de relaxation.

Deuxièmement, sur le graphique de droite, nous avons représenté les différences $u_1^1 - u_2^1$ et $u_1^2 - u_2^2$ qui représentent la contrainte couplante et qui doivent donc être nulles afin de respecter celle-ci et donc avoir une solution réalisable. Nous observons alors que seul l'algorithme de décomposition par prédiction `seq` respecte la contrainte couplante. Pour les 2 autres, ils la respectent seulement lorsqu'ils ont convergé.

Pour l'algorithme séquentiel `seq2`, si l'on veut respecter la contrainte couplante à chaque itération, il suffit de prendre à l'itération k, $U_{N+1}^{(k)}$ et $U_1^{(k)}, ..., U_N^{(k)}$. Cela se traduit par le fait de récupérer la solution de N premiers sous-problèmes de l'itération précédente. Nous devons réaliser cette manipulation car la contrainte d'allocation du sous-problème N+1 d'une itération est associée à la somme des solutions des autres sous-problèmes de l'itération précédente.

Enfin, pour l'algorithme parallèle, nous pouvons réaliser la même opération.

Nous avons donc testé différents types de décomposition afin de résoudre le problème de réseaux de distribution d'eau connectés. Nous avons pu voir les avantages et désavantages de chaque décomposition.

Voici un tableau récapitulant tous les éléments pour chaque méthode :

Décomposition	**Nombre d'itérations**	**Contrainte couplante**	**Ensemble admissible**
Prix	faible	NON	OUI
Quantités	élevé	OUI	NON (Blocage)
Quantités2	élevé	OUI	OUI
Prédiction `par`	élevé	OUI	OUI
Prédiction `seq`/`seq2`	moyen	OUI	OUI

TABLE 1 – Résumé de chaque décompositon

Dans ce tableau, nous comparons le nombre d'itérations mais pas la vitesse. Nous nous intéresserons à la vitesse dans la dernière partie.

Nous avons ainsi dans ces 3 premières parties comparé les différentes méthodes de décomposition. Nous allons maintenant voir plus en détails les différences entre une programmation séquentielle et parallèle avec les algorithmes de prédiction.

2.4 Comparaison d'une programmation séquentielle et parallèle

Dans cette nouvelle sous-partie, nous allons comparer les différents algorithmes séquentiels et parallèle avec différentes expériences.

Premièrement, nous allons nous intéresser à l'influence des paramètres de relaxation α et γ sur l'exemple jouet avec une précision de 10^{-3}. Pour commencer, nous allons tester les 3 algorithmes de prédiction en ne réalisant une relaxation ni sur l'allocation ni sur les prix, c'est à dire $\gamma = 1$ et $\beta = 1$.

Nous obtenons les résultats suivants :
- `par` : $u_1 = u_2 = \begin{pmatrix} 1 & 0.156 \end{pmatrix}$ `et` $v_1 = \begin{pmatrix} 0.469 & 0.375 \end{pmatrix}$
- `seq` : $u_1 = u_2 = \begin{pmatrix} 1 & 0.156 \end{pmatrix}$ `et` $v_1 = \begin{pmatrix} 0.469 & 0.375 \end{pmatrix}$
- `seq2` : $u_1 = u_2 = \begin{pmatrix} 0.750 & 0.938 \end{pmatrix}$ `et` $v_1 = \begin{pmatrix} 0.250 & 0.063 \end{pmatrix}$

Les résultats ne sont pas du tout bons, aucun des algorithmes n'a convergé vers la bonne solution. De plus, nous remarquons que chacun des algorithmes a atteint son nombre maximum d'itérations donc aucun des algorithmes n'a convergé.

Afin de voir de plus près pourquoi les algorithmes ne convergent pas, nous allons représenter la solution de u_1 à chaque itération des algorithmes. Voici les graphiques :

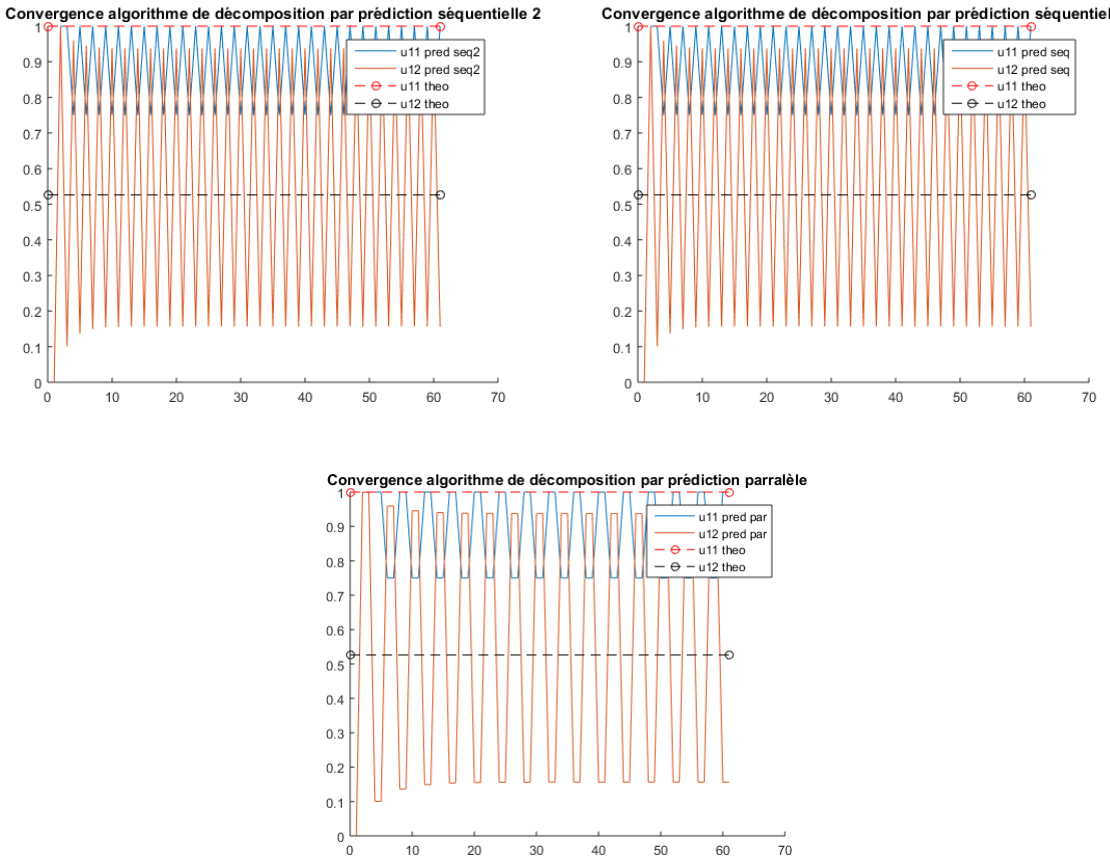

FIGURE 16 – Analyse résolution exemple jouet par la décomposition par prédiction sans aucune relaxation

Nous observons alors que sans relaxation les algorithmes alternent entre 2 solutions et n'arrivent jamais à converger. Nous ne pourrons alors jamais choisir $\gamma = 1$ et $\beta = 1$.

Nous allons maintenant regarder ce qui se passe lorsque nous réalisons une relaxation.

Tout d'abord, notre objectif est de ne pas réaliser de relaxation sur l'allocation. En effet, si nous réalisons une relaxation sur l'allocation alors la contrainte couplante n'est plus respectée à chaque itération et par conséquent nous ne gardons plus l'avantage de la décomposition par quantités avec la décomposition par prédiction.

Pour illustrer ce fait, nous allons réaliser une relaxation sur l'allocation avec l'algorithme `seq`. Nous avons vu tout à l'heure que lorsque nous ne réalisions pas de relaxation sur l'allocation, celui-ci respectait la contrainte couplante. Nous allons donc pouvoir comparer. Nous prenons $\gamma = 1$ et $\beta = 0.5$.

Nous obtenons les résultats suivants : $u_1 = u_2 = \begin{pmatrix} 1 & 0.526 \end{pmatrix}$ `et` $v_1 = \begin{pmatrix} 0.263 & 0.211 \end{pmatrix}$.

L'algorithme converge bien vers la bonne solution en 11 itérations.

Voici les graphes résumant l'expérience :

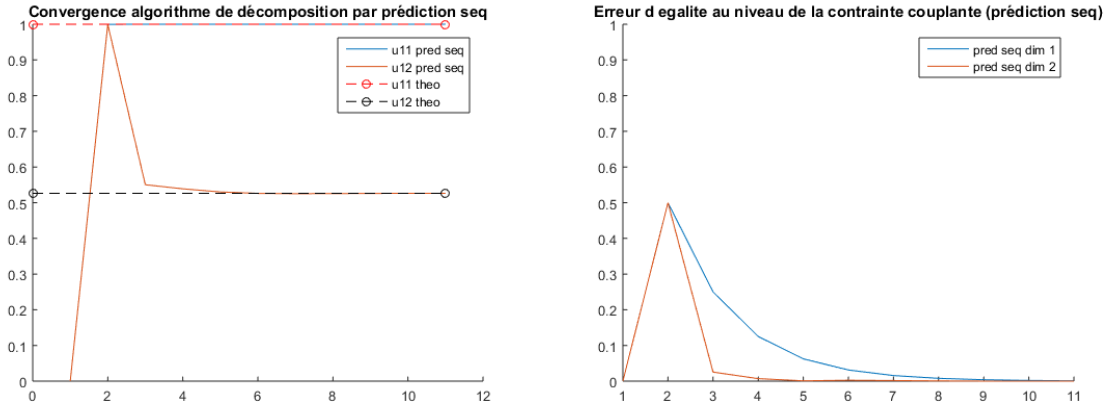

FIGURE 17 – Analyse résolution exemple jouet par la décomposition par prédiction `seq` avec relaxation sur les allocations

Nous observons alors bien que l'algorithme converge mais que la contrainte couplante n'est pas respectée à chaque itération.

En réalisant une relaxation sur l'allocation, nous perdons donc un peu de l'intérêt de la décomposition par prédiction par rapport à la décomposition par les prix. Il faut donc essayer d'éviter au maximum de réaliser une relaxation sur l'allocation. Néanmoins, en choisissant la valeur de β, nous pouvons jauger à quel point nous voulons que la contrainte couplante soit respectée tout au long de l'algorithme.

Enfin, pour analyser l'influence de β et γ, nous réalisons une analyse du nombre d'itérations de l'algorithme en fonction de ces dernières :

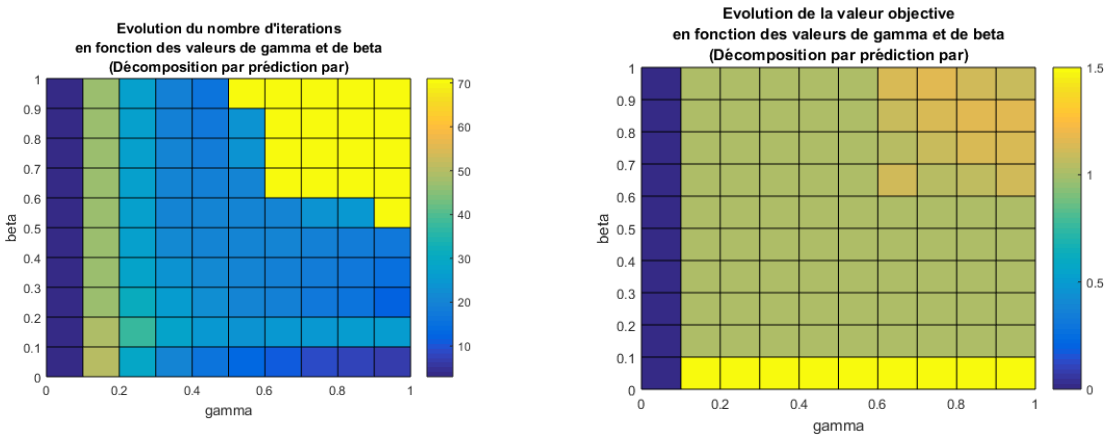

FIGURE 18 – Analyse résolution exemple jouet par la décomposition par prédiction `par` en fonction de α et γ

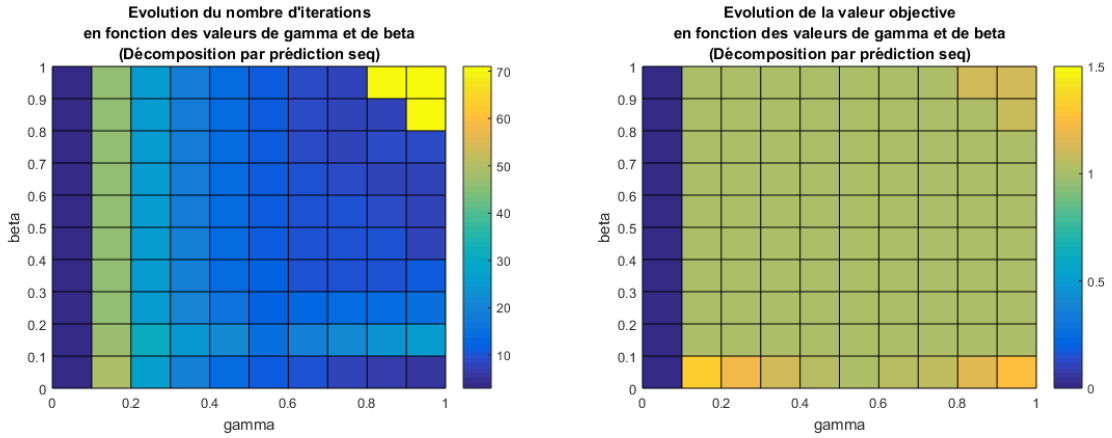

FIGURE 19 – Analyse résolution exemple jouet par la décomposition par prédiction `seq` en fonction de α et γ

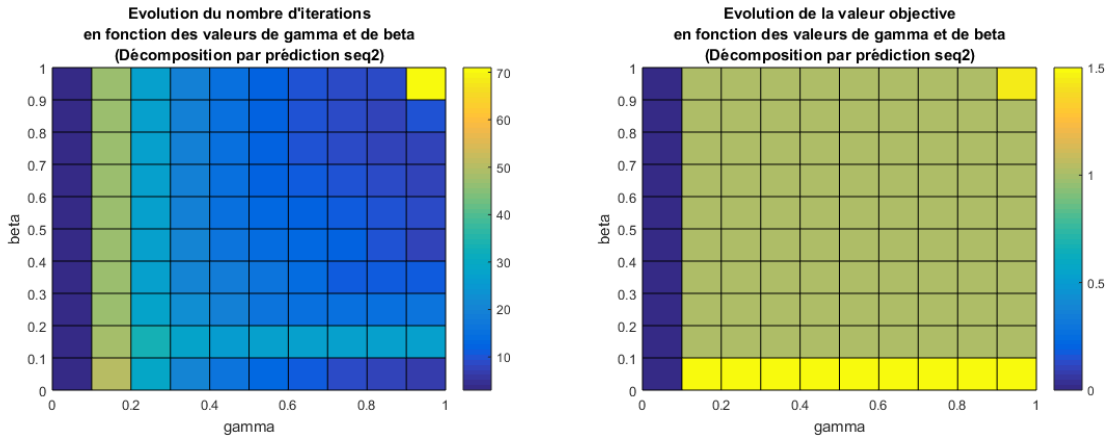

FIGURE 20 – Analyse résolution exemple jouet par la décomposition par prédiction `seq2` en fonction de α et γ

Pour chacune des méthodes, nous pouvons observer le nombre d'itérations ainsi que la valeur de la fonction objectif en fonction de γ et β. Lorsqu'une cellule est jaune, cela signifie que l'algorithme a atteint le nombre d'itération maximale fixé égal à 70. Il faut faire aussi attention à vérifier si le nombre d'itérations est faible, si la valeur objective est bien faible aussi. Ici, nous ne nous intéressons pas aux valeurs avec $\gamma = 0$ car la contrainte couplante n'est pas prise en compte.

Premièrement, pour l'algorithme parallèle, nous observons qu'il existe des valeurs de β lorsqu'on fixe $\gamma = 1$ tel que l'algorithme converge. Nous pouvons donc garder le respect de la contrainte couplante. Nous arrivons à obtenir un nombre minimal d'itération de 12 pour $\gamma = 1$ et $\beta = 0.2$. C'est une bonne nouvelle car dans ce cas la contrainte couplante est respectée à chaque itération.

Deuxièmement, pour l'algorithme séquentiel `seq`, nous observons qu'avec quasiment toutes les valeurs de β et γ, l'algorithme converge à part lorsqu'on essaye de fixer les 2 proches de 1.

Nous arrivons à obtenir un nombre minimal d'itération de 7 pour $\gamma = 1$ et $\beta = 0.5$. Encore une fois la contrainte couplante est respectée et c'est une bonne nouvelle.

Enfin, pour l'algorithme séquentiel `seq2`, nous observons aussi qu'avec quasiment toutes les valeurs de β et γ à part (1,1), l'algorithme converge. Nous arrivons à obtenir un nombre minimal d'itérations de 7 pour $\gamma = 1$ et $\beta = 0.5$. Encore une fois la contrainte couplante est respectée et c'est une bonne nouvelle.

Nous observons alors que les paramètres γ et β influent beaucoup sur le nombre d'itérations. Nous avons déterminé des valeurs optimales pour chaque algorithme. Les valeurs de γ et β vont aussi avoir une influence sur la contrainte couplante. Nous avons vu que pour que la contrainte couplante soit respectée, nous devons fixer $\gamma = 1$. Dans notre cas, cela tombe bien car les valeurs optimales de β et γ donnent $\gamma = 1$ pour tous les algorithmes.

Les algorithmes séquentiels convergent avec moins d'itérations que l'algorithme parallèle. Nous allons donc maintenant aussi comparer le temps d'exécution pour différentes valeurs de N.

Pour cela, nous créons une fonction sur MATLAB qui crée des instances en fonction de certains paramètres afin de fixer les valeurs maximums et minimums de α,β, a et b. Voici nos résultats :

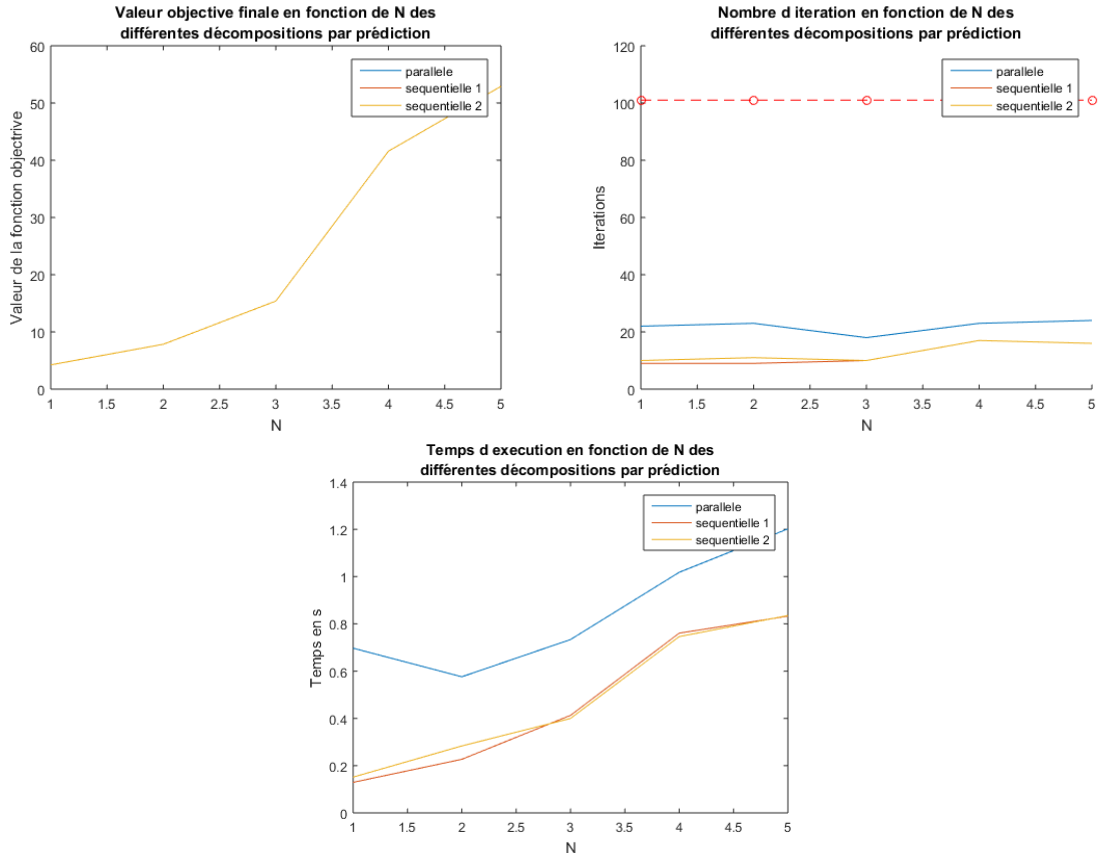

FIGURE 21 – Analyse résolution en fonction de N

Sur ces graphes, nous pouvons observer le nombre d'itérations, le temps d'exécution et la valeur de la fonction objectif finale pour les 3 algorithmes avec des problèmes dont la valeur de N va de 1 à 5.

Premièrement, sur le premier graphe, nous observons que les 3 valeurs objectives finales sont les mêmes. Les trois algorithmes ont donc bien convergé pour les différentes valeur de N.

Deuxièmement, sur le graphe à droite, nous observons que les algorithmes séquentielles ont besoin de moins d'itérations pour converger que l'algorithme parallèle. Nous avions déjà vu cela précédemment. Nous pouvons aussi noter que le nombre d'itérations est quasiment le même entre les deux algorithmes séquentiels.

Dernièrement, nous observons le même phénomène pour le temps d'exécution. Les algorithmes séquentiels convergent plus vite que l'algorithme parallèle.

Dans cette partie, nous avons donc vu que les algorithmes séquentielles sont meilleurs que l'algorithme parallèle dans notre problème, il est donc inutile d'utiliser un algorithme parallèle.

Dans la dernière sous-partie, nous allons comparer les performances des algorithmes de décomposition en fonction des différents paramètres. Pour la décomposition par prédiction, nous n'utiliserons que l'algorithme prédiction `seq`. En effet, il est inutile d'utiliser l'algorithme parallèle. De plus, l'algorithme `seq2` s'exécute de la même manière que `seq` mais ne respecte que la contrainte couplante en réalisant la petite manipulation décrite précédemment.

2.5 Étude de performance des algorithmes

Dans cette partie, nous allons analyser l'influence des pas et des précisions ainsi que de N. Nous n'utiliserons que la méthode des points intérieurs ou la formule explicite car ce sont les méthodes les plus rapides avec de grands N.

Premièrement, nous allons commencer par l'influence du pas sur le temps d'exécution et le nombre d'itérations. La décomposition par prédiction ne possède pas de pas de gradient donc nous nous intéressons ici qu'aux décompositions par prix et quantités.

Nous nous plaçons avec $N = 10$ et une précision de 10^{-2}. Nous calculons alors le temps d'exécution et le nombre d'itérations pour différentes valeurs de ρ, le pas de gradient.

Voici les graphiques que nous obtenons :

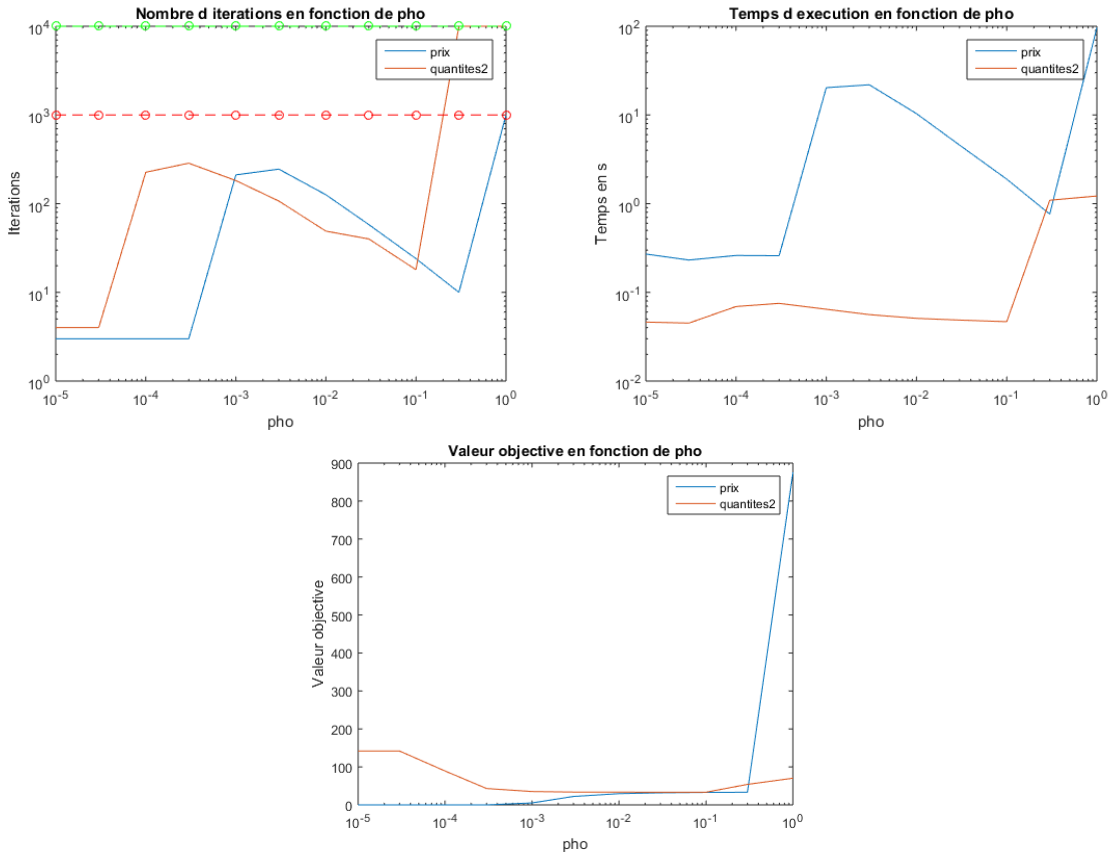

FIGURE 22 – Analyse résolution en fonction du pas

Premièrement, nous pouvons observer qu'avec des pas très petits l'algorithme converge vers une mauvaise solution en peu d'itérations. En effet, le pas est tellement petit que la valeur de u change peu et donc l'algorithme s'arrête. Nous observons cela sur le dernier graphe avec la valeur objective en fonction de ρ : la valeur objective pour la décomposition par quantités est haute et la valeur objective de la décomposition par prix est très basse car la contrainte couplante n'est pas du tout respectée pour le moment. Nous observons le même phénomène pour des pas trop grands avec des valeurs qui explosent cette fois.

Deuxièmement, nous observons que :

- pour les prix, l'algorithme arrive au mieux à converger en 10 itérations et 0.76 secondes pour $\rho = 0.3$.
- pour les quantités, l'algorithme arrive au mieux à converger en 18 itérations et 0.048 secondes pour $\rho = 0.1$.

L'algorithme par les prix nécessite donc moins d'itérations mais est beaucoup plus long. Ceci s'explique par l'utilisation des formules explicites pour l'algorithme de décomposition par les quantités.

Nous allons maintenant étudier l'influence de N sur le temps d'exécution et le nombre d'itérations avec une précision de 10^{-2}. Nous fixons les pas de gradient à une valeur un peu en dessous de leur valeur optimal afin d'éviter que le pas soit très grand pour certaine instance aléatoire.

Voici les graphiques que nous obtenons :

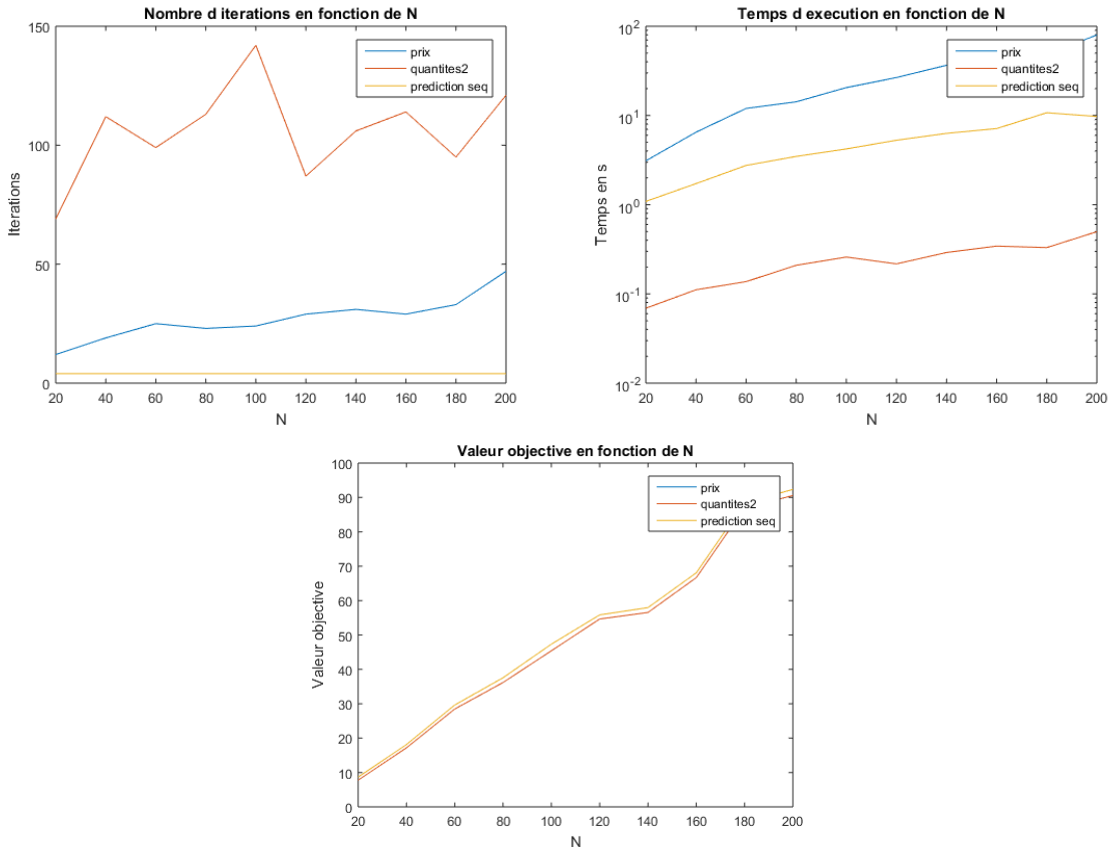

FIGURE 23 – Analyse résolution en fonction du pas

Premièrement, sur le graphique de gauche, nous observons que les algorithmes de prédiction et de prix convergent avec moins d'itérations que quantités. Nous observons aussi que le nombre d'itérations des décomposition par prédiction et quantités ne dependent pas vraiment de N contrairement à l'algorithme de décomposition par prix.

Néanmoins, sur le graphique de droite, nous observons que les temps d'exécution des trois algorithmes augmentent avec N. La décomposition par quantités est l'algorithme le plus rapide après nous avons la prédiction puis les prix. La résolution des sous-problèmes par une formule explicite est vraiment efficace.

Enfin, sur le dernier graphe, nous observons que la décomposition par prédiction ne donne pas de bons résultats. En effet, nous avions vu sur le premier graphique qu'elle convergeait très rapidement en 4 itérations à chaque fois, mais sa valeur optimale est à chaque fois supérieure

aux algorithmes de décomposition par prix et quantités. Si nous voulons une meilleure valeur optimale, il faut donc augmenter la précision et donc sûrement augmenter le nombre d'itérations et le temps d'exécution.

Pour terminer sur ce problème, nous allons étudier l'influence de la précision sur le temps d'exécution et le nombre d'itérations avec N=10. Nous fixons les pas de gradient à une valeur un peu en dessous de leur valeur optimal afin d'éviter que le pas soit très grand pour certaines instances aléatoires.

Voici les graphiques que nous obtenons :

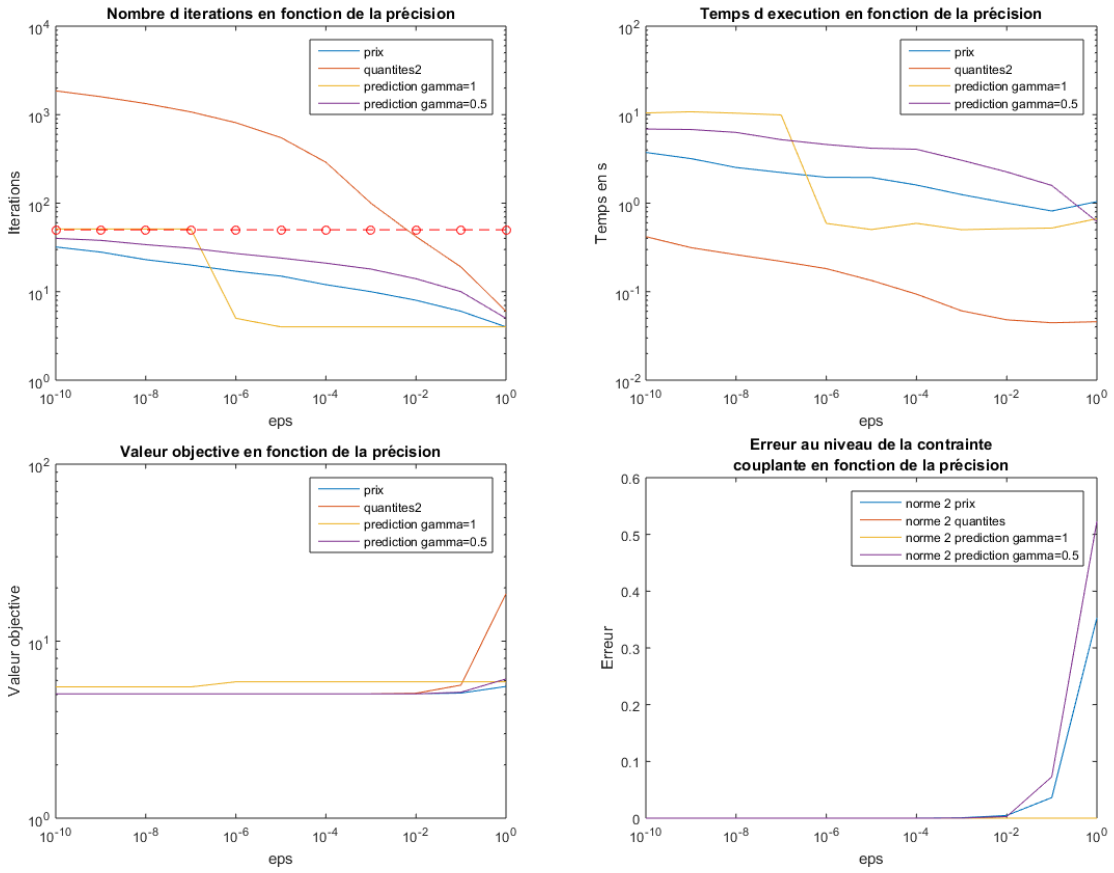

FIGURE 24 – Analyse résolution en fonction de la précision

Nous pouvons alors observer que le nombre d'itérations et le temps augmentent avec la précision pour toutes les méthodes.

En revanche, nous avons dû tester 2 méthodes de prédiction. En effet, à partir d'une précision de 10^{-7}, la décomposition par prédiction n'arrivait plus à converger avec $\gamma = 1$, elle atteint le nombre maximal d'itérations. Nous étions trop à la limite de la convergence. Il a donc fallu diminuer γ et réaliser une relaxation. Nous aurions pu aussi diminuer β. Nous observons aussi que la valeur de la fonction objectif est plus haute que les autres avec la prédiction et $\gamma = 1$ comme avec l'étude en fonction de N.

Enfin, nous observons que lorsque la précision est faible (10^{-6}), la décomposition par prix et prédiction possède des valeurs objectives plus faibles que la décomposition par quantités. La valeur peut même être inférieure à la valeur optimale même si ici ce n'est pas le cas. Néanmoins, cela s'explique par le fait que la contrainte couplante n'est pas tout à fait respectée comme le montre le graphe en bas à droite.

Pour conclure, lors de cet exercice, nous avons testé différentes méthodes de décomposition. Nous avons vu leurs avantages et leurs inconvénients. Nous avons aussi étudié les différences entre une programmation parallèle et séquentielle. Enfin, nous avons étudié les performances de chacun des algorithmes. Dans notre cas, l'algorithme de décomposition par quantités 2 semble le plus performant et cohérent pour résoudre le problème.

Exercice 3 : Optimisation d'un portefeuille d'actions et le modèle Markowitz

On suppose que l'on possède N actions, que l'on représente par des variable aléatoires $(X_i)_{i=1...N}$. Chaque action rapporte en moyenne à l'actionnaire $e_i = \mathbb{E}(X_i)$ au bout d'un an. On suppose que l'on investit une somme S donnée, et l'on note $u_i \in \mathbb{R}$ la proportion de la somme investie dans l'action i. Le portefeuille total est représenté par la variable aléatoire : $X = \sum_{i=1}^{N} u_i X_i$ et rapporte ainsi en moyenne $\mathbb{E}(X) = \sum_{i=1}^{N} u_i e_i$.

Dans son modèle Markowitz utilise les 2 premiers moments de la variable à étudier, qui sont l'espérance et la variance, correspondant respectivement au rendement global espéré et au risque encouru : $\theta = \mathbb{E}(X) = \langle e, u \rangle$ et $\sigma^2 = var(X) = \langle u, Qu \rangle$, où $u := (u_i)_{i=1...N}$ et Q la matrice des covariances.

On peut soit imposer un rendement donné $R_e > 0$, i.e., $\theta \geq R_e$ soit fixer une dose de risque donnée $D_e > 0$, i.e., $\sigma^2 \leq De$. On peut modéliser le risque du portefeuille par les deux problèmes quadratiques suivants :

Problème 1 :

$$\begin{aligned} \min_{u} \quad & \frac{1}{2}\langle u, Qu \rangle \\ \text{subject to} \quad & \sum_{i=1}^{N} u_i e_i \geq R_e \\ & \sum_{i=1}^{N} u_i = 1 \end{aligned} \tag{3.1}$$

Ce problème revient à minimiser le risque associé aux achats des actions tout en respectant le rendement R_e imposé.

Problème 2 :

$$\begin{aligned} \min_{u} \quad & \sum_{i=1}^{N} -u_i e_i \qquad \text{(équivalent à maximiser)} \\ \text{subject to} \quad & \langle u, Qu \rangle \leq D_e \\ & \sum_{i=1}^{N} u_i = 1 \end{aligned} \tag{3.2}$$

Ce problème revient à maximiser le rendement de l'achat des actions tout en ayant un risque inférieur à la dose D_e imposée.

Ces algorithmes ne sont pas décomposables dans le cas où les actions ne sont pas indépendantes i.e Q n'est pas diagonale. Pour les résoudre nous utiliserons la méthode du problème auxiliaire. De plus, nous ajoutons la contrainte d'interdire la vente à découvert, ce qui revient à forcer la positivité des parts u_i à investir.

Afin d'obtenir un problème d'optimisation convexe on considérera une matrice Q définie positive. Q est toujours symétrique car correspond à une matrice de variance-covariance.

3.1 Formulation du problème auxiliaire pour le problème (3.1)

Décomposition de l'objectif Pour ce problème l'objectif n'est pas décomposable, on le sépare donc en sa partie additive (A) et couplante (C), i.e $J(U) = J^A(U) + J^C(U)$. On définit alors :

$$\begin{cases} Q_A := \begin{pmatrix} * & & (0) \\ & \ddots & \\ (0) & & * \end{pmatrix} \\ Q_C := \begin{pmatrix} 0 & & (*) \\ & \ddots & \\ (*) & & 0 \end{pmatrix} \end{cases} \quad \text{telles que} \quad \begin{cases} Q = Q_A + Q_C \\ J^A(U) = \frac{1}{2}\langle U, Q_A U\rangle \\ J^C(U) = \frac{1}{2}\langle U, Q_C U\rangle \end{cases}$$

La matrice de covariance Q_C étant symétrique, on a $\nabla J^C(U) = Q_C U$

Décomposition des contraintes On écrit les contraintes sous leur forme standard et on obtient

$$\forall i = 1 \dots N \qquad \theta_i(u_i) = \begin{pmatrix} -e_i \\ 1 \end{pmatrix} u_i \qquad U_i^{ad} = \{u \in \mathbb{R} \,|\, u \geq 0\} = \mathbb{R}_+ \quad \text{et} \quad v = \begin{pmatrix} -R_e \\ 1 \end{pmatrix}$$

Écriture du problème auxiliaire On récupère la formulation générale pour l'écriture du problème auxiliaire. Afin de s'assurer de la convergence on ajoute le terme auxiliaire $A^{(k)}$. Le problème devient dans notre cas :

$$\min_{u \in U^{ad}} \quad A^{(k)}(u) + \varepsilon_k J^A(u) + \langle \varepsilon_k Q_C u^k, u\rangle + \varepsilon_k \langle \theta(u), p^{(k)}\rangle \qquad (\mathbb{PPA}1)$$

Ce nouveau problème est décomposable. On résout ses sous-problèmes par la méthode d'Uzawa présentée en introduction $(*)$.
Remarque : On considère dans notre implémentation $A^{(k)}(u) := \frac{\alpha_k}{2}\|u\|^2 - \alpha_k\langle u^k, u\rangle$ *(ce terme est fortement convexe et vérifie* $\nabla A^{(k)}(u^{(k)}) = 0$*).*

Coordination : Actualisation des prix Le problème présente à la fois des contraintes d'égalité et d'inégalité, ce qui donne la formule d'actualisation suivante :

$$\begin{cases} p_1^{(k+1)} = \max\left(0, p_1^{(k)} + \beta\varepsilon_k \left(\sum_{i=1}^{N} -e_i u_i^k + R_e\right)\right) \\ p_2^{(k+1)} = p_2^{(k)} + \beta\varepsilon_k \left(\sum_{i=1}^{N} u_i^k - 1\right) \end{cases}$$

Algorithme On continue d'itérer l'algorithme jusqu'à ce que $\frac{\left\|u^{(k+1)} - u^{(k)}\right\|^2}{\left\|u^{(k)}\right\|^2} + \frac{\left\|p^{(k+1)} - p^{(k)}\right\|^2}{\left\|p^{(k)}\right\|^2} < \epsilon$.

3.2 Implémentation et résultats de la formulation ($\mathbb{PPA}1$)

Afin de vérifier nos algorithmes nous les comparerons aux solutions obtenues par la fonction `fmincon` de MATLAB pour des exemples simples en petite dimension.

Test de l'algorithme sur des instances de petite taille

On considère comme exemples les problèmes suivant avec $N = 2$ et $N = 3$ et on visualise les solutions données par l'algorithme associé au PPA.

Exemple 1 :

$$Q = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \quad e = \begin{pmatrix} 4 \\ 5 \end{pmatrix} \quad Re = \frac{9}{2} \quad \text{ayant pour solution optimale} \quad u^* = \frac{1}{2}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

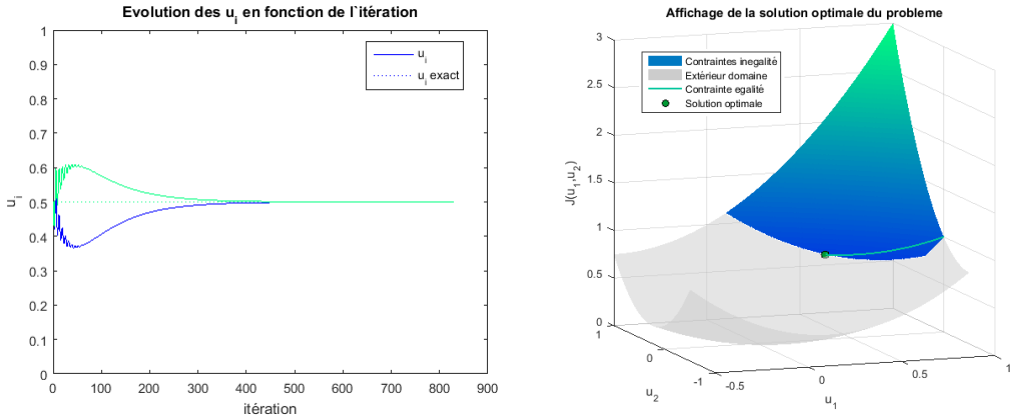

FIGURE 25 – Convergence vers la solution exacte et affichage de la solution pour l'exemple 1

Exemple 2 :

$$Q = \begin{pmatrix} 3 & 1 & 0 \\ 1 & 5 & 1 \\ 0 & 1 & 2 \end{pmatrix} \quad e = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \quad Re = 1 \quad \text{ayant pour solution optimale} \quad u^* = \frac{1}{21}\begin{pmatrix} 8 \\ 1 \\ 12 \end{pmatrix}$$

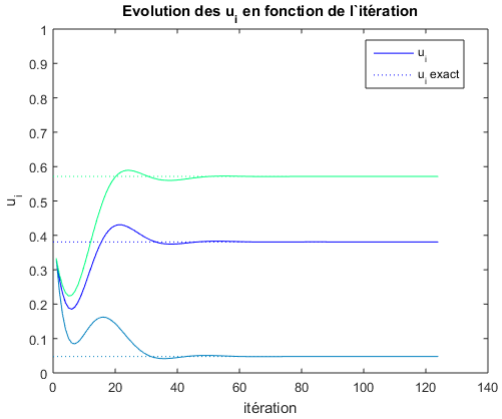

FIGURE 26 – Convergence vers la solution exacte pour l'exemple 2

Le graphe en 3 dimensions présenté à la figure 25 permet de rendre compte de la forme quadratique de l'objectif et de la forme linéaire des contraintes d'inégalité et d'égalité. Pour $N = 2$, ce problème revient donc à minimiser un polynôme de degré 2 sur un segment de droite.

On observe que l'algorithme proposé converge bien vers la solution exacte pour les exemples proposés en un nombre d'itérations limité et dans des temps d'exécution proches de 0.1s pour une précision $\epsilon = 10^{-6}$. Cela nous permet de valider numériquement notre algorithme. Il reste cependant moins efficace que le solveur natif de MATLAB, nous le testons maintenant sur des instances de plus grande taille.

Résolution d'un problème de grande taille

On considère maintenant un exemple pour un problème de taille $N = 20$. On définit pour cela une matrice Q symétrique et définie positive. On obtient les résultats suivants :

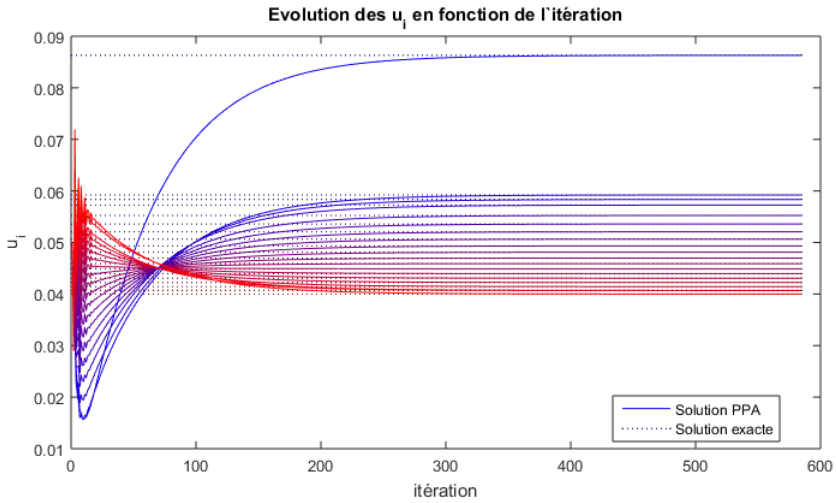

FIGURE 27 – Convergence vers la solution exacte pour une instance de grande taille

Le graphe précédent permet de traduire l'évolution de la solution à mesure des itérations. On observe ainsi les interactions entre les différentes unités. Malgré la taille de l'instance le solveur natif reste plus rapide. La vitesse de convergence est directement liée au nombre d'itérations et dépend du choix des hyper-paramètres qui sera discuté en dernière partie.

3.3 Formulation du problème auxiliaire pour le problème (3.2)

Décomposition de l'objectif dans ce problème l'objectif est décomposable et on peut directement écrire $J(u) = J^A(u) = -\langle e, u\rangle$.

Décomposition des contraintes Les contraintes ne sont pas décomposables et présentent des termes couplants. On les réécrit sous la forme : $\theta(u) = \theta^A(u) + \theta^C(u)$ où θ^A correspond aux termes additifs et θ^C aux termes couplants. On définit les matrices Q_A et Q_C de manière analogue au problème précédent. On obtient alors :

$$U_i^{ad} = \{U \in \mathbb{R} \,|\, u \geq 0\} = \mathbb{R}_+ \qquad v = \begin{pmatrix} D_e \\ 1 \end{pmatrix}$$

$$\theta^A(u) = \textstyle\sum_{i=1}^N \theta_i^A(u_i) \quad \text{avec} \quad \theta_i^A(u_i) = \begin{pmatrix} Q_{ii}u_i^2 \\ u_i \end{pmatrix}$$

$$\theta^C(u) = \begin{pmatrix} \langle Q_C u, u\rangle \\ 0 \end{pmatrix} \quad \text{avec} \quad \nabla\theta^C(u) = \begin{pmatrix} 2Q_C u \\ 0 \end{pmatrix}$$

Écriture du problème auxiliaire On écrit le problème auxiliaire à partir de la formulation générale du cours. Le terme $A^{(k)}$ est le même terme auxiliaire que pour le problème précédent.

$$\min_u \quad A^{(k)}(u) - \varepsilon_k\langle e, u\rangle + \varepsilon_k\left\langle \begin{pmatrix} \langle 2Q_C u^k, u\rangle \\ 0 \end{pmatrix} + \sum_{i=1}^N \begin{pmatrix} Q_{ii}u_i^2 \\ u_i \end{pmatrix}, p^{(k)} \right\rangle \tag{$\mathbb{PPA}2$}$$

Ce nouveau problème est décomposable. On résout ses sous-problèmes par la méthode d'Uzawa.

Coordination : Actualisation des prix On procède de la même manière que précédemment.

$$\begin{cases} p_1^{(k+1)} = \max\left(0, p_1^{(k)} + \beta\varepsilon_k\left(\langle 2Qu^k, u^k\rangle - D_e\right)\right) \\ p_2^{(k+1)} = p_2^{(k)} + \beta\varepsilon_k\left(\sum_{i=1}^N u_i^k - 1\right) \end{cases}$$

Algorithme On arrête l'algorithme lorsque $\frac{\left\|u^{(k+1)}-u^{(k)}\right\|^2}{\left\|u^{(k)}\right\|^2} + \frac{\left\|p^{(k+1)}-p^{(k)}\right\|^2}{\left\|p^{(k)}\right\|^2} < \epsilon$.

3.4 Implémentation et résultats de la formulation ($\mathbb{PPA}2$)

De manière analogue au premier problème, on teste l'algorithme sur des exemples simples avant de l'appliquer à des instances plus grandes.

Test de l'algorithme sur des instances de petite taille

On considère comme exemples les problèmes suivants avec $N = 2$ et on visualise les résultats obtenus pour chacun d'eux :

Exemple 1 :

$$Q = \begin{pmatrix} 3/2 & 0 \\ 0 & 2 \end{pmatrix} \quad e = \begin{pmatrix} 3 \\ 2 \end{pmatrix} \quad De = 1 \quad \text{ayant pour solution optimale} \quad u^* = \frac{1}{7}\begin{pmatrix} 4+\sqrt{2} \\ 3-\sqrt{2} \end{pmatrix}$$

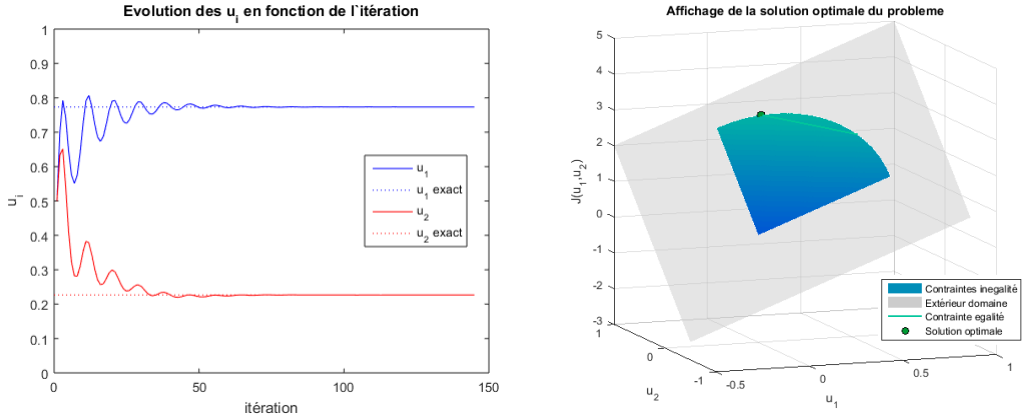

FIGURE 28 – Convergence vers la solution exacte et affichage de la solution pour l'exemple 1

Exemple 2 :

$$Q = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \quad e = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad De = \frac{3}{2} \quad \text{ayant pour solution optimale} \quad u^* = \frac{1}{\sqrt{2}}\begin{pmatrix} \sqrt{2}-1 \\ 1 \end{pmatrix}$$

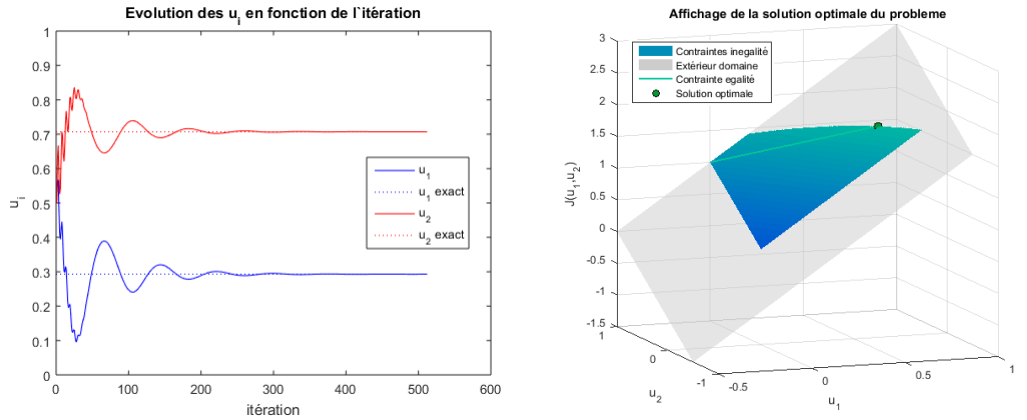

FIGURE 29 – Convergence vers la solution exacte et affichage de la solution pour l'exemple 2

On observe sur les figures 28 et 29 que, contrairement au problème précédent, l'objectif est linéaire et la contrainte sur le risque est quadratique. Les solutions obtenues convergent bien vers la solution exacte. Ces tests permettent ainsi de valider numériquement l'algorithme implémenté.

Enfin, on observe que l'algorithme converge en un nombre faible d'itérations et en moins de 0.1s pour les exemples ci-dessus. On le compare au solveur MATLAB pour des instances de plus grande taille.

Résolution d'un problème de grande taille

On considère désormais un problème de taille $N = 20$ que l'on résout à l'aide de l'algorithme précédent et le solveur MATLAB.

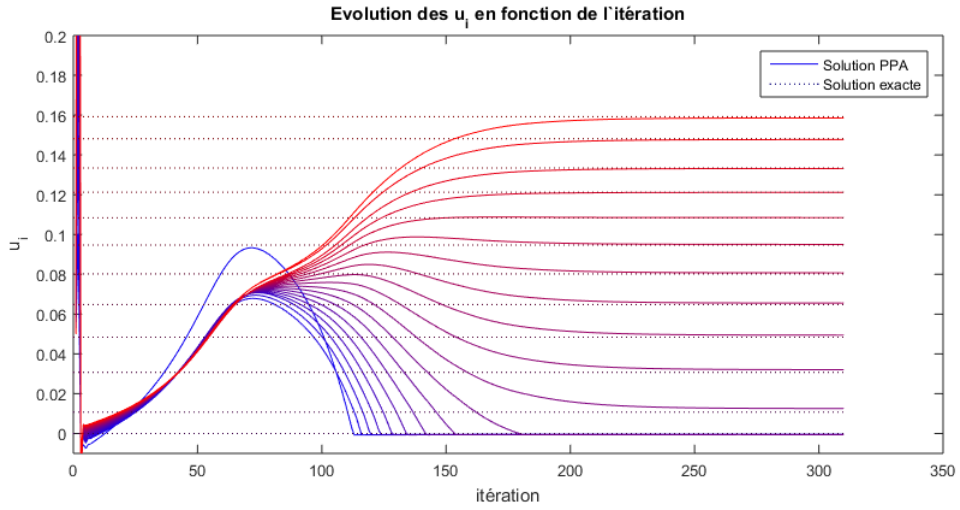

FIGURE 30 – Convergence vers la solution exacte pour une instance de grande taille

Les résultats des deux algorithmes sont les mêmes, l'algorithme PPA résout le problème en 310 itérations. Le solveur MATLAB reste plus rapide malgré l'augmentation de la dimension, la vitesse de convergence dépend du choix des hyper-paramètres et du seuil de précision choisi qui était $\epsilon = 10^{-6}$ dans notre cas.

3.5 Discussion sur les résultats des algorithmes

Choix du terme auxiliaire

Pour conclure, les décompositions obtenues diffèrent de celles des autres exercices car les problèmes ne sont plus additifs. Ces décompositions reposent sur l'ajout du terme auxiliaire fortement convexe $A^{(k)}(u)$ et la qualité de convergence des algorithmes dépend des hyper-paramètres qui lui sont associés. Nous avons choisi $A^{(k)}(u) := \frac{\alpha_k}{2} \|u\|^2 - \alpha_k \langle u^k, u \rangle$, ce terme vérifie de plus la condition $\nabla A^{(k)}(u^{(k)}) = 0$.

Choix des hyper-paramètres

Afin de limiter le nombre d'itérations des algorithmes $\mathbb{PPA}1$ et $\mathbb{PPA}2$ il a été important d'ajuster les hyper-paramètres α_k, ε_k associés au terme auxiliaire et β associé à l'actualisation du prix. Ils peuvent être interprétés de la manière suivante :

— α_k traduit la force de la convexité du terme auxiliaire et doit être positif.

— ε_k est associé au problème initial et doit vérifier

$$\exists \gamma > 0 \text{ tels que, } \forall k \in \mathbb{N}\, 0 < \varepsilon_k < \frac{2r^{(k)}}{M + \gamma} \qquad \begin{cases} A^{(k)} \text{ est convexe de module } r(k) \\ \nabla J \text{ est Lipschitzienne de constante } M \end{cases}$$

Exercice 4 : Optimisation de la gestion de smart grids

Intéressons-nous à présent à un problème de gestion de réseau électrique à grande échelle. Un réseau est composé de différents agents producteurs ou consommateurs de puissance qui interagissent entre eux (les échanges de puissance et les communications entre les différents agents constituent un smart grid). L'objectif du problème est d'atteindre l'équilibre des puissances produites et consommées. Pour répartir correctement les puissances demandées et produites aux multiples agents sur des situations réelles, il convient de pouvoir traiter en peu de temps des problèmes d'optimisation de grande taille (i.e avec un grand nombre d'agents). Enfin, lorsque l'on traite ce genre de problème, il faut tout de même garder une certaine flexibilité pour prendre en compte les sources d'énergies renouvelables qui figurent parmi les producteurs car ces dernières sont souvent sources d'incertitudes.

On note désormais N le nombre d'agents et P_i la puissance injectée dans le réseau (si $P_i > 0$ alors c'est une production tandis que si $P_i < 0$ c'est une consommation) par chaque agent i (pour $i = 1, \dots, N$). Chaque agent i possède une fonction de coût f_i dépendant de sa puissance consommée ou produite P_i (fonction que nous détaillerons par la suite) ainsi qu'une limite maximale de production de puissance notée $P_{max,i}$ (cette limite de production sera naturellement fixée à 0 pour les consommateurs de puissance). Il est aussi possible de considérer des limites minimales de production de puissance $P_{min,i}$ dans ce problème mais nous avons fait le choix de ne pas en tenir compte par la suite ($\forall i = 1, \dots, N$, $P_{min,i} = -\infty$). Enfin, chaque agent possède une puissance idéale $P_{0,i}$ qu'il doit fournir au réseau.

On s'intéresse désormais au problème formulé de la manière suivante :

$$\begin{aligned} \min_{P=(P_i)_{1\leq i\leq N}} \quad & J(P) = \sum_{i=1}^{N} f_i(P_i) \\ \text{subject to} \quad & \sum_{i=1}^{N} P_i = 0 \\ & P_i \leq P_{max,i} \qquad \forall i \in \{1, \dots, N\} \end{aligned} \tag{4}$$

Dans le cadre de ce projet, la fonction de coût f_i de chaque agent aura la forme polynomiale suivante :

$$f_i(P_i) = a_i(P_i - P_{0,i})^2 + b_i$$

La variable P_i représente la puissance injectée dans le réseau de l'agent i et le coefficient positif a_i traduit la pénibilité de s'éloigner de la puissance idéale (plus a_i est grand, plus l'écart entre P_i et $P_{0,i}$ sera facturé cher). Enfin, b_i représente le coût de base lié à la puissance idéale de l'agent i. La fonction ci-dessus traduit donc un coût relatif à la puissance idéalement fournie par l'agent correspondant.

Remarquons dans un premier temps que problème formulé ci-dessus est un problème d'optimisation quadratique convexe. En effet, la fonction objectif est une somme de fonctions quadratiques convexes $f_i(P_i)$ (car chaque coefficient a_i est strictement positif), elle est donc convexe. On peut aussi écrire la fonction objectif sous la forme $J(P) = \frac{1}{2}\langle P, AP\rangle - \langle P, b\rangle$ et remarquer que $A = diag(a_1, ..., a_N)$. Comme $a_i > 0\ \forall i \in \{1, ..., N\}$, la matrice A est symétrique définie positive donc J est fortement convexe. De plus, la contrainte liant les variables P_i est linéaire et l'ensemble admissible de chaque variable U_i^{ad} est fermé et convexe. Par ailleurs, le fait que l'ensemble admissible soit fermé (l'intersection des ensemble U_i^{ad} est fermée) et que la fonction objectif J soit continue et coercive (fonction quadratique convexe) nous assure qu'il y a bien existence d'une solution au problème. De plus, le fait que J soit convexe et que l'ensemble admissible soit convexe nous assure l'unicité de cette solution.

On remarque ensuite que ce problème est décomposable car il possède une structure additive. On le met désormais sous la forme :

$$\begin{array}{ll} \min\limits_{\{P_i \in U_i^{ad}\}_{i=1\dots N}} & \sum\limits_{i=1}^{N} J_i(P_i) \\ \text{s. t} & \sum\limits_{i=1}^{N} \theta_i(P_i) = 0 \end{array} \quad \text{avec } \forall i = 1 \dots N \begin{cases} J_i(P_i) = a_i(P_i - P_{0,i})^2 + b_i \\ \theta_i(P_i) = P_i \\ U_i^{ad} = \{P \in \mathbb{R} \,|\, P \leq P_{max,i}\} \end{cases}$$

Cette structure additive ainsi que les différentes hypothèses citées précédemment concernant le problème (problème quadratique convexe admettant une unique solution) vont nous permettre de lui appliquer des méthodes de décomposition/coordination telles que les algorithmes de décomposition par les prix et de décomposition par les quantités (par allocation). Ces dernières sont explicitées par la suite dans le cadre de ce problème.

4.1 Décomposition par les prix

Voici le détail des 2 étapes de l'algorithme de décomposition par les prix à chaque itération k appliqué au problème décrit jusqu'ici. On suppose que l'on connaît une valeur initiale de prix $p^{(1)}$.

Décomposition À prix $p^{(k)}$ fixé, on résout les N sous problèmes $\mathbb{P}_i^k$ (pour $i = 1, ..., N$) afin d'obtenir les valeurs de P_i^{k+1}.

$$\forall i = 1, \dots, N \qquad \min_{u_i \in U_i^{ad}} a_i(P_i - P_{0,i})^2 + b_i + \langle p^{(k)}, P_i\rangle \qquad (\mathbb{P}_i^k)$$

Coordination On met à jour le prix de la manière suivante (avec $\rho_k > 0$) pour obtenir de nouvelles valeurs $p^{(k+1)}$:

$$p^{(k+1)} = p^{(k)} + \rho_k \left(\sum_{i=1}^{N} P_i^{k+1} \right)$$

Algorithme On applique l'algorithme de décomposition par les prix (3).

4.2 Décomposition par les quantités

Voici le détail des 2 étapes de l'algorithme de décomposition par les quantités à chaque itération k appliqué au problème décrit jusqu'ici. On suppose que l'on connaît des valeurs initiales d'allocation $\omega_i^{(1)}$ satisfaisant $\sum_{i=1}^{N} \omega_i^1 = 0$.

Décomposition À allocations $\omega_i^{(k)}$ fixées, on résout les N sous problèmes $\mathbb{Q}_i^k$ (pour $i = 1, ..., N$) afin d'obtenir les valeurs de P_i^{k+1} et les multiplicateurs de Lagrange λ_i^{k+1} (associés à la contrainte locale de chaque sous-problème).

$$\forall i = 1, \ldots, N \quad \begin{cases} \min\limits_{P_i \in U_i^{ad}} & a_i(P_i - P_{0,i})^2 + b_i \\ \text{s.t} & P_i = \omega_i^{(k)} \end{cases} \tag{$\mathbb{Q}_i^k$}$$

Coordination On met à jour les allocations de la manière suivante (avec $\rho_k > 0$) pour obtenir des nouvelles valeurs $\omega_i^{(k+1)}$ ($\forall i = 1, ..., N$) :

$$\omega_i^{(k+1)} = \omega_i^{(k)} + \rho_k \left(\lambda_i^{k+1} - \frac{1}{N} \sum_{j=1}^{N} \lambda_j^{k+1} \right)$$

Algorithme On applique l'algorithme de décomposition par les quantités (4).

Nous allons maintenant appliquer ces 2 algorithmes sur une instance du problème possédant $N = 15$ agents dont les caractéristiques sont détaillées dans les tableaux suivants (table 2 et table 3) :

Producteur	**Nombre**	P_0	a	b	$P_{max}(MW)$	**Capacité (MW)**
Centrale à charbon	1	8.5	10	1000	10	10
Éolienne	5	1.9	1	10	2	2
Barrage réversible	1	3	0.1	100	16	16
Panneau photovoltaïque	2	1.9	0.9	11	2	2

TABLE 2 – Données des producteurs

Consommateurs	**Nombre**	P_0	a	b	$P_{max}(MW)$	**Capacité (MW)**
DataCenter	1	-10	5	20	0	10
Logement ($\times$ 7500)	1	-7.5	1	10	0	7.5
Usine	1	-9	5	8	0	9
Tramway	2	-0.12	6	9	0	0.12
Hôpital	1	-0.2	200	30	0	0.2

TABLE 3 – Données des consommateurs

La situation décrite par cette instance est représentée par la figure 31. Nous supposerons enfin que le réseau électrique est considéré comme parfait (avec des plaques de cuivre idéales). Ainsi, nous négligerons les pertes.

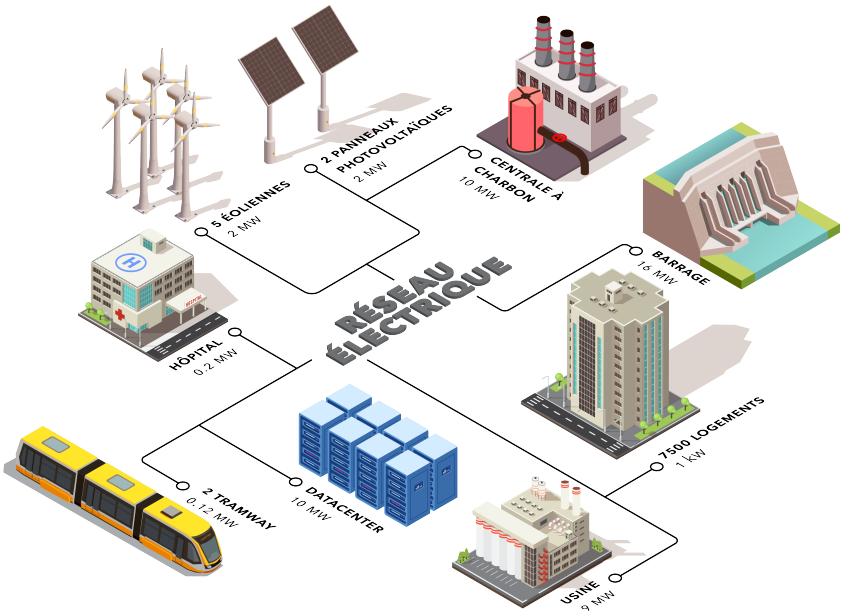

FIGURE 31 – Schéma du réseau électrique

De plus, nous considérerons les 2 scénarios suivants :

— **Scénario n°1** : Tous les producteurs cités précédemment fonctionnent normalement (situation de base).

— **Scénario n°2** : On considère qu'il n'y a plus de vent et donc que les éoliennes ne fonctionnent plus ; donc la puissance produite est diminuée. On fixe donc $P_{max} = 0$ pour les éoliennes (diminution des énergies renouvelables).

Les algorithmes de décomposition par les prix et par les quantités ont été appliqués sur ces 2 scénarios avec ces données. L'implémentation a été réalisée ur MATLAB. Pour chaque cas, nous avons fixé une précision ε à 10^{-4} et utilisé des pas ρ_k constants pour les étapes de coordination. L'algorithme d'Uzawa a été utilisé pour résoudre les sous-problèmes dans la phase de coordination également avec une précision fixée ε à 10^{-4}. Dans ce cas d'étude, les sous-problèmes des 2 algorithmes sont également quadratiques convexes et satisfont les mêmes hypothèses que le problème global (critère continu, coercif, convexe + ensemble admissible fermé et convexe), il y a donc existence et unicité de la solution. Lors des différents tests que nous avons effectués, nous avons à la fois fait varier le pas global ρ de l'algorithme de décomposition et le pas de l'algorithme d'Uzawa ρ_{UZAWA}. Lorsque l'on applique l'algorithme d'Uzawa à un problème quadratique sous la forme $\min_{u \in K} J(u) = \frac{1}{2}\langle Au, u\rangle - \langle b, u\rangle$ avec $K := \{u \in \mathcal{R}, Cu \leq c, Du = d\}$, on sait que la suite des solutions $u^{(k)}$ converge vers la solution optimale u^* si $0 < \rho_{UZAWA} < \rho_c$ avec ρ_c donné par $\rho_c = \frac{2.vp_{min}(A)}{\|C\|^2 + \|D\|^2}$. Dans l'algorithme de décomposition par les prix, pour chaque sous-problème $i = 1, ..., N$, $\rho_c = 4a_i$, nous avons donc choisi ρ_{UZAWA} tel

que $\rho_{UZAWA} < 4min_{\{i=1,\dots,N\}} a_i$. De la même manière, dans l'algorithme de décomposition par les quantités, $\rho_c = 2a_i$, donc on choisit ρ_{UZAWA} tel que $\rho_{UZAWA} < 2min_{\{i=1,\dots,N\}} a_i$.

Pour chaque scénario, nous avons d'abord cherché la solution exacte du problème à l'aide d'un solveur sur MATLAB puis nous l'avons comparée à nos résultats obtenus avec les 2 algorithmes. Les résultats de nos différentes expérimentations sont présentées dans les 2 sections suivantes. Nous avons regroupé dans des tableaux, les valeurs de la solution renvoyée par le solveur P_{opt} **- solveur** (solution exacte qui nous sert de référence), les valeurs de la solution obtenue avec l'algorithme de décomposition par les prix P_{opt} **- prix**, les valeurs de la solution obtenue avec l'algorithme de décomposition par les quantités P_{opt} **- quant**, les puissances maximales de chaque agent $P_{max}(MW)$ (qui définissent les ensembles admissibles des solutions), les valeurs de coût de chaque solution, les valeurs du multiplicateur de Lagrange associé à la contrainte $\sum_{i=1}^{N} P_i = 0$ (dans les conditions de KKT), les valeurs de ρ et de ρ_{UZAWA} utilisées, le temps d'exécution (en secondes) et le nombre d'itérations.

4.3 Scénario n° 1

On résout le problème (4) énoncé plus haut par les méthodes de décomposition-coordination (prix et quantités). Les différents résultats figurent dans la table 4. De plus on représente graphiquement l'évolution des valeurs de la solution et des valeurs du multiplicateur en fonction des itérations pour chaque algorithme (figure 32 et figure 33).

Agent	P_{opt} **- solveur**	P_{opt} **- prix**	P_{opt} **- quant**	$P_{max}(MW)$
Centrale à charbon	8.5122	8.5121	8.508	10
Éolienne	2	2.0002	1.9962	2
Barrage réversible	4.2166	4.2149	4.2129	16
Panneau photovoltaïque	2	2.0002	1.9963	2
DataCenter	-9.9757	-9.9757	-9.9799	0
Logement ($\times$ 7500)	-7.3783	-7.3785	-7.3822	0
Usine	-8.9757	-8.9757	-8.9799	0
Tramway	-0.0997	-0.0998	-0.1039	0
Hôpital	-0.1994	-0.1994	-0.1986	0
Coût	1258.2432	1258.243	1258.2322	-
Multiplicateur de Lagrange	-0.24332	-0.243	-0.24331	-
ρ	-	0.05	0.1	-
ρ_{UZAWA}	-	0.1	0.01	-
Temps (s)	-	0.1655	3.1761	-
Nombre d'itérations	-	54	885	-

TABLE 4 – Résultats scénario 1

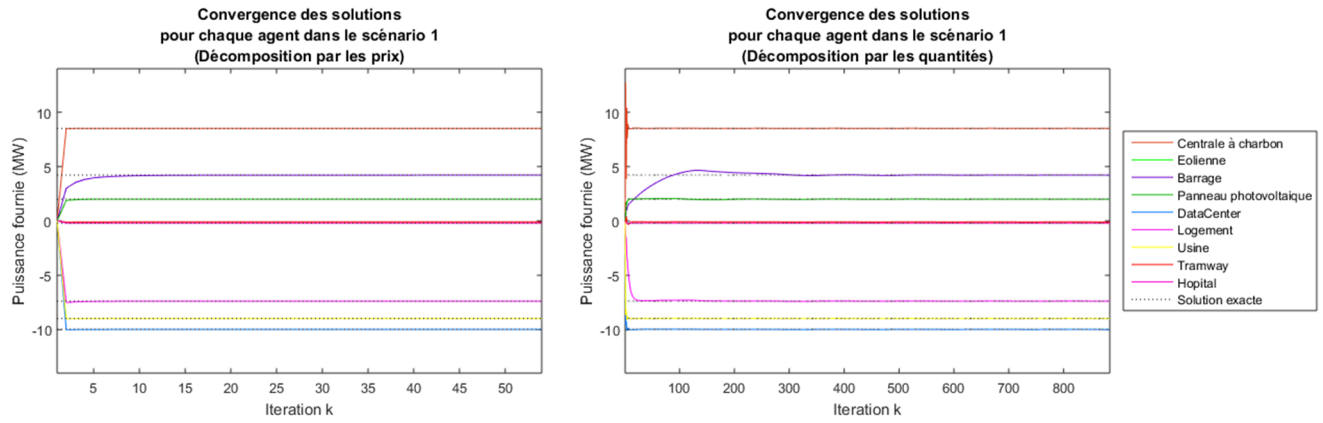

FIGURE 32 – Illustration de la convergence des solutions (Scénario 1)

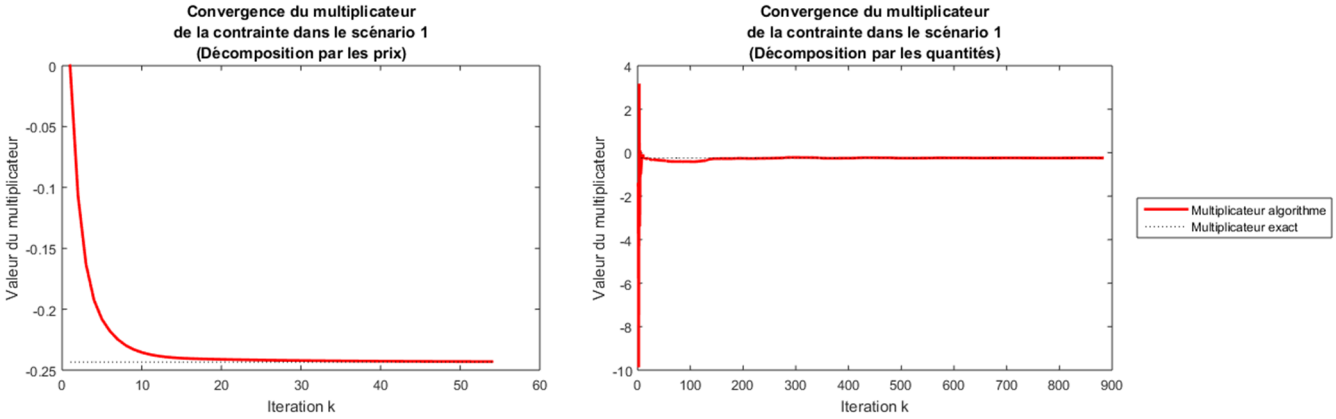

FIGURE 33 – Illustration de la convergence du multiplicateur (Scénario 1)

On peut d'abord noter que les valeurs des solutions ainsi que les valeurs du multiplicateur obtenues avec les algorithmes de décomposition par les prix et de décomposition par les quantités sont très proches de celles obtenues avec le solveur. Les implémentations des 2 algorithmes renvoient donc les bonnes solutions pour ce problème. La figure 32 confirme graphiquement que les solutions convergent vers les valeurs attendues. De la même manière, la figure 33 montre que les valeurs du multiplicateur se rapprochent de la valeur théorique au fil des itérations. On peut remarquer que pour chaque agent, les valeurs de puissance fournies sont bien inférieures à leurs valeurs de P_{max} respectives (avec une précision de 10^{-4}), donc les solutions obtenues respectent bien numériquement les contraintes liées à l'ensemble admissible. La valeur obtenue pour la somme des puissances des agents est de l'ordre de 10^{-4} pour la solution générée par l'algorithme de décomposition par les prix et de 10^{-2} pour la solution obtenue avec l'algorithme de décomposition par les quantités ; donc la contrainte $\sum_{i=1}^{N} P_i = 0$ est bien respectée numériquement. Enfin, on peut observer que les valeurs optimales renvoyées par les 2 algorithmes sont

bien les mêmes et elles correspondent également à la valeur donnée par le solveur. On remarque également que l'algorithme de décomposition par les prix a convergé plus vite et en moins d'itérations que l'algorithme de décomposition par les quantités (pour ce jeu de paramètres donné et pour ce problème en particulier).

4.4 Scénario n° 2

On résout le même problème dans le cas où les éoliennes ne peuvent pas produire d'énergie par manque de vent. Ceci revient à fixer $P_{max} = 0$ pour chacune des 5 éoliennes. Les différents résultats figurent dans la table 5. Comme pour le scénario 1, on représente graphiquement l'évolution des valeurs de la solution et des valeurs du multiplicateur en fonction des itérations pour chaque algorithme (figure 34 et figure 35).

Agent	P_{opt} **- solveur**	P_{opt} **- prix**	P_{opt} **- quant**	$P_{max}(MW)$
Centrale à charbon	8.5973	8.5973	8.6239	10
Éolienne	0	0	-0.0078	0
Barrage réversible	12.7349	12.7299	12.7395	16
Panneau photovoltaïque	2	2.0001	1.9999	2
DataCenter	-9.8053	-9.8054	-9.8094	0
Logement ($\times$ 7500)	-6.5265	-6.527	-6.5306	0
Usine	-8.8053	-8.8054	-8.8094	0
Tramway	0	-0.0024	0.0192	0
Hôpital	-0.1951	-0.1951	-0.1965	0
Coût	1287.1439	1287.1378	1287.3945	-
Multiplicateur de Lagrange	-1.947	-1.946	-1.946	-
ρ	-	0.1	0.01	-
ρ_{UZAWA}	-	0.05	0.001	-
Temps (s)	-	0.3661	20.0472	-
Nombre d'itérations	-	122	7053	-

Table 5 – Résultats scénario 2

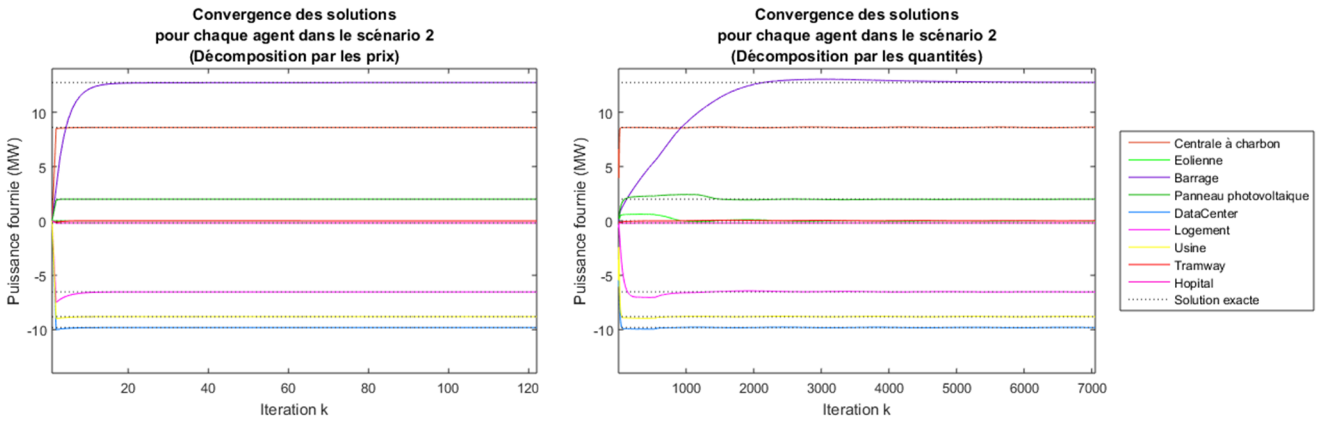

FIGURE 34 – Illustration de la convergence des solutions (Scénario 2)

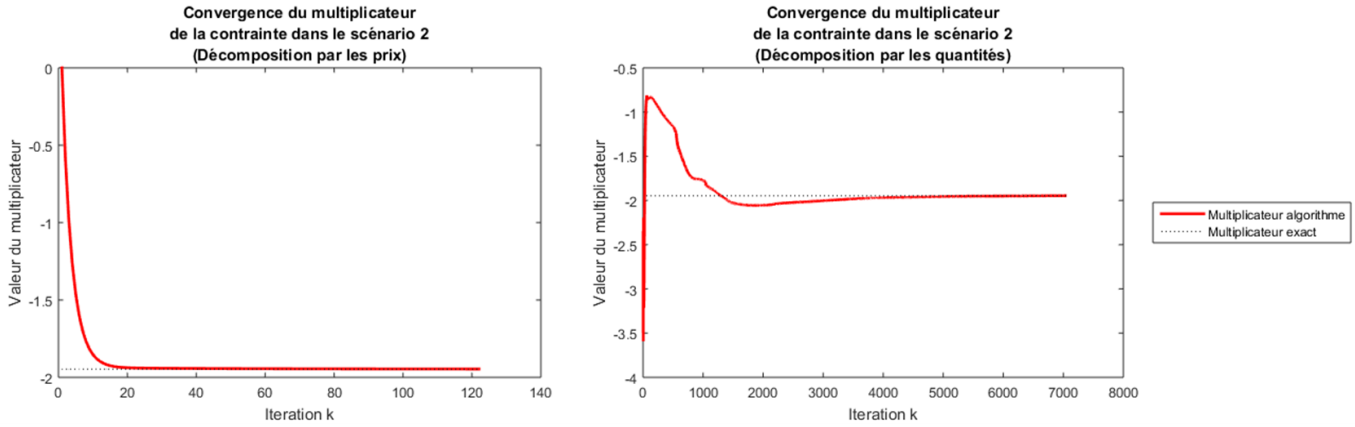

FIGURE 35 – Illustration de la convergence du multiplicateur (Scénario 2)

De la même manière, on peut observer sur ces résultats que nos implémentations des 2 algorithmes (de décomposition par les prix et par les quantités) renvoient des solutions et une valeur optimale très proche de celles renvoyées par le solveur. On observe également graphiquement que les solutions et que les valeurs du multiplicateur convergent vers les valeurs attendues sur les figures 34 et 35. Les solutions obtenues respectent toutes les contraintes relatives à P_{max} avec une précision de 10^{-4} et satisfont donc numériquement les contraintes liées à l'ensemble admissible. La somme des puissances des agents obtenue par l'algorithme de décomposition par les prix est de l'ordre de 10^{-4} tandis que celle obtenue avec l'algorithme de décomposition par les quantités est de l'ordre de 10^{-2}. La contrainte $\sum_{i=1}^{N} P_i = 0$ est donc numériquement vérifiée. Enfin, comme pour le scénario précédent, nous pouvons noter que l'algorithme de la décomposition par les prix a convergé plus vite et en moins d'itérations que l'algorithme de décomposition par les quantités (pour cet ensemble de paramètres et pour ce problème).

4.5 Bilan et discussion sur les résultats des algorithmes

Sur les résultats présentés ci-dessus, nous avons vu que les implémentations des algorithmes de décomposition par les prix et de décomposition par les quantités avaient convergé vers des solutions très proches de la solution exacte (obtenue avec le solveur) du problème dans les configurations des 2 scénarios et ceci en un temps raisonnable (en au plus 20 secondes dans le pire des cas). Les solutions obtenues respectaient bien numériquement les contraintes, et les valeurs optimales correspondantes étaient également très proches de la valeur théorique attendue.

Les résultats ont été présentés avec des jeux de paramètres fixés (notamment pour les valeurs de ρ et de ρ_{UZAWA}) mais nous avons pu observer qu'en changeant les valeurs des pas, les algorithmes convergeaient toujours vers ces mêmes solutions avec d'éventuels écarts dont l'ordre de grandeur n'excédait pas 10^{-2}. Les valeurs des solutions obtenues sont donc peu sensibles au choix des paramètres (à condition que ces derniers soient suffisamment petits pour qu'il respectent les conditions de convergence). En revanche, les valeurs de ces derniers ont une grande influence sur le temps d'exécution et le nombre d'itérations pour converger (les valeurs utilisées pour présenter nos résultats ont été choisies afin d'avoir des temps d'exécution raisonnables). Enfin, au cours de nos expérimentations, nous avons pu observer que l'algorithme de décomposition par les prix convergeait plus vite que l'algorithme de décomposition par les quantités. En effet, d'une manière générale, la méthode de décomposition par les prix est plus facile à mettre en place que la décomposition par les quantités (sur ce problème, on pouvait par exemple noter que les sous-problèmes de la décomposition par les prix ne devaient satisfaire qu'une seule contrainte tandis que ceux de la décomposition par les quantités devaient en satisfaire 2). Cependant, la décomposition par les quantités fournit des solutions qui respectent les contraintes à chaque itération alors que ce n'est pas le cas pour la décomposition par les prix. Ainsi, malgré son exécution plus lente, la méthode de décomposition par les quantités peut quand même fournir une solution admissible si l'algorithme s'arrête prématurément (bien que celle-ci ne soit pas forcément optimale).