Méthodes numériques

On distingue trois types de méthodes numériques en commande optimale : les méthodes directes, les méthodes indirectes, et enfin les méthodes basées sur la résolution des équations de Hamilton-Jacobi-Bellman (HJB). Les méthodes directes consistent à discrétiser l'état et le contrôle, ramenant le problème à un problème d'optimisation non-linéaire. Les méthodes indirectes consistent à résoudre numériquement, par une méthode de tir, un problème en appliquant le principe de Pontryagin. Enfin, les méthodes basées sur la résolution des équations HJB peuvent s'appliquer soit directement, soit en utilisant une résolution séquentielle, grâce à la programmation dynamique.

Dans ce chapitre, nous présentons quelques-unes des méthodes numériques communément utilisées pour le type de problèmes qui sont traités dans ce mémoire. Nous tenterons de montrer les équivalences entre les méthodes, lorsqu'elles existent, autant d'un point de vue numérique que théorique.

Enfin, un algorithme de tir original, nommé SCOP, est présenté. Il permet de résoudre des problèmes du type (1.1) mais n'est pas applicable, à l'heure actuelle, sur des problèmes du type (1.2).

Dans le cas de l'optimisation de la répartition de puissance d'un véhicule hybride sur cycle prédéfini, on se trouve dans le cadre de problèmes de type (1.1) et (1.2). La fonction L correspond à la consommation instantanée de carburant, la commande u est soumise à des contraintes de bornes dépendantes du temps, correspondant aux saturations des différents actionneurs.

Dans un soucis de clarté, les méthodes seront écrites pour une résolution de problèmes du type (1.1), dans lequel la contrainte dure sur $x(T)$ est remplacée par une fonction de pénalisation ϕ, comme pour le problème (PO) (équations (2.1)). Ce problème s'écrit

$$(PO_{appl}) \begin{cases} \min\limits_{u \in U} \left\{ J(u) = \displaystyle\int_0^T L(u(t),t)dt + \phi(x(T),T) \right\} \\ \text{avec :} \\ \dot{x} = f(u(t),t), \quad x(0) = x_0 \\ x_{\min} \leq x(t) \leq x_{\max} \\ u_{\min}(t) \leq u(t) \leq u_{\max}(t), \end{cases} \tag{3.1}$$

Cependant, dans le cas des problèmes du type (1.2), la programmation dynamique a été appliquée, les méthodes indirectes n'étant pas utilisables à cause de la présence de l'état discret.

3.1 Méthode directe : optimisation non-linéaire sous contraintes

La méthode directe est la méthode la plus évidente lorsque l'on doit résoudre un problème de commande optimale. En discrétisant l'état et la commande dans le problème (3.1), on se ramène à un problème d'optimisation non-linéaire en dimension finie (N variables) :

$$\begin{cases} \min\limits_{u_k \in U_k} J(u) := \sum\limits_{k=0}^{N-1} L_k(u_k)\Delta t + \phi(x_N) \\ \text{avec :} \quad x_{k+1} = f_k(x_k, u_k), \quad x(0) = x_0. \\ x_{\min} \leq x(t) \leq x_{\max} \\ u_{\min}(t) \leq u(t) \leq u_{\max}(t), \end{cases} \tag{3.2}$$

où $x_{k+1} = f_k(x_k, u_k)$ représente la version discrète de l'équation d'état $\dot{x} = f(u(t), t)$, discrétisée en utilisant par exemple la méthode d'Euler explicite.

Pour résoudre ce problème, on suppose que le temps est subdivisé de manière égale, tel que $0 = t_0 < t_1 < ... < t_N = T$, le pas de discrétisation étant noté Δt. On suppose que la commande reste constante par morceau durant le pas de temps Δt. Les contraintes sur la commande ou sur l'état sont appliquées sur les valeurs discrétisées.

3.1.1 Résolution numérique

La résolution de ce problème peut s'effectuer par exemple via un algorithme de type SQP (Sequential Quadratic Programming), [Nocedal and Wright, 2000].

Cette méthode a été implémentée, mais la mise en oeuvre du problème et la prise en compte des contraintes est assez complexe et contraignante, et le temps nécessaire à la convergence est important. Pour ces raisons, la programmation non-linéaire n'a pas été retenue pour l'étude paramétrique présentée dans la section 4.

3.1.2 Paramétrisation simplifiée du contrôle (découpage par zones)

Dans la précédente méthode, les paramètres du contrôle u sont les $u_k, k = 0, ..., N-1$. Au lieu de rechercher les N commandes optimales, problème d'optimisation complexe si N est élevé, on peut imposer des règles pour la définition de u, règles définies par des paramètres dont le nombre serait $N_r \ll N$. La difficulté est alors de trouver empiriquement des règles intuitives permettant de définir la commande u d'une façon simple, à l'aide d'un nombre restreint de paramètres.

Dans l'application étudiée, cela pourrait revenir à définir des seuils de couples et/ou de régime au-dessus desquels certaines valeurs de u seraient figées.

Plutôt que d'optimiser une commande en fonction du temps, on peut imposer des zones de fonctionnement (voir [Voise et al., 2005]) pour le moteur électrique, ainsi que pour le fonctionnement simultané des deux moteurs, etc. Dans un second temps, une loi de gestion d'énergie propre à chaque zone pourra être définie, permettant une utilisation de cette stratégie en temps-réel. Cette paramétrisation permet de réduire grandement la taille du problème d'optimisation, puisque seules quelques variables devront être utilisées dans la définition du problème d'optimisation.

Cette technique ne peut se faire sans connaissance du système : on définit, par exemple, une commande $\tilde{u}$ telle que :

$$\tilde{u} = \begin{cases} u_1 \text{ si } T_{rq} < T_a, \\ u_2 \text{ si } T_a \leq T_{rq} < T_b, \\ u_3 \text{ si } T_b \leq T_{rq}. \end{cases} \tag{3.3}$$

La stratégie obtenue, après optimisation, reste évidemment sous-optimale, puisque la paramétrisation enlève des degrés de liberté à la commande.

3.2 Méthodes de tir

Les méthodes de tir consistent à résoudre directement les conditions d'optimalité du problème (PO). Elles sont basées sur le Principe de Pontryagin. Ces méthodes sont notamment très utilisées dans le contrôle des véhicules spatiaux, où la précision est primordiale. Elles ont pour avantage d'être numériquement très précises, et relativement rapides comparées aux méthodes indirectes.

3.2.1 Méthode de tir simple

La méthode de tir consiste à itérer sur la valeur de p pour atteindre une valeur objectif. On définit une fonction de tir, notée Γ, telle que

$$\Gamma(p_0) = p(T) - \phi'(x(T), T). \tag{3.4}$$

La résolution des conditions d'optimalité revient alors à ajuster la valeur de p_0 telle que

$$\Gamma(p_0) = 0. \tag{3.5}$$

En pratique, pour résoudre ce type de problème, on utilisera une méthode numérique itérative, qui consiste à itérer sur la valeur de p_0 lorsque l'on possède une expression analytique pour les équations (2.3b) et (2.3d).

Cette méthode convient donc bien à la résolution de problèmes dont la forme est analytique, permettant de calculer une forme elle aussi analytique pour l'expression de u^*. Lorsque la valeur initiale du paramètre p_0 est bien choisie, sa convergence est assez rapide. Néanmoins, cette méthode n'est plus valide en présence de contraintes actives sur l'état. En effet, l'état adjoint p ne suit plus la loi d'évolution définie par (2.3b) lorsque l'état x touche ponctuellement ou longe une contrainte, puisqu'un nouveau multiplicateur associé à la contrainte sur l'état doit être introduit.

3.2.2 Méthode de tir multiple

Dans certains problèmes d'optimisation, le système différentiel régissant l'évolution des variables d'état n'est pas le même au cours de la trajectoire.

L'évolution des variables d'état pour ce type de problème s'écrit par exemple :

$$(\dot{x}(t), \dot{p}(t)) = \begin{cases} f_0(t, x(t), p(t)) & \text{si } 0 \leq t \leq t_1 \\ f_1(t, x(t), p(t)) & \text{si } t_1 \leq t \leq t_2 \\ \vdots & \\ f_N(t, x(t), p(t)) & \text{si } t_N \leq t \leq T, \end{cases} \tag{3.6}$$

x étant l'état, p étant l'état adjoint, et $t_1, t_2, ...t_N \in [0, T]$ étant des temps de commutation.

Lorsqu'il est possible de dériver une expression analytique de la commande optimale, la méthode de tir multiple ([Bryson and Ho, 1975], [Evans, 2000]) se trouve être bien adaptée. Elle consiste à résoudre

le problème d'optimisation comme dans la méthode de tir simple, avec des conditions de continuité (ou conditions de jonction) sur les variables d'état et d'état adjoint, lors d'un changement du système différentiel.

Ce type d'algorithme se trouve donc bien adapté à la résolution de problèmes d'optimisation avec contraintes d'état. En effet, dans ce cas les temps $t_1, t_2, ...t_N \in [0,T]$ peuvent être des temps de jonction avec un arc frontière (contrainte sur l'état active pendant un temps donné), ou bien des temps de contact avec la frontière (l'état touche une contrainte, mais la quitte aussitôt).

Un très bon exemple se trouve dans [Bonnans and Hermant, 2008], exemple qui est repris dans [Rousseau et al., 2008c], voir section (3.4.6) : il s'agit de connaître la trajectoire optimale d'une corde élastique attachée à ses deux extrémités, et qui repose sur un support plan. Lorsque la corde ne touche pas le support, l'équation représentant la trajectoire de la corde est parfaitement connue. En revanche, lorsque la corde touche le support, il devient difficile de connaître sa trajectoire, notamment à quels endroits elle touche le support puis le quitte.

On peut alors "découper" la trajectoire en 3 tronçons, chaque partie étant décrite par une équation différentielle différente. Les temps de commutation représentent alors les abscisses auxquels la corde touche puis quitte le support. Dans certains problèmes très simples, on peut résoudre le problème, en écrivant les conditions d'optimalité associées. Différents cas sont énumérés dans [Bryson and Ho, 1975] : système continu avec état final fixé à un temps donné, état final fixé à un temps final libre (incluant les problèmes à temps minimal), etc. L'auteur résout les problèmes posés en adjoignant l'équation d'état et les contraintes associées à l'état, à la fonction à minimiser, puis définit les conditions d'optimalité associées au problème via une méthode de perturbation (telle que celle qui est utilisée dans (2.8)). Le lecteur pourra aussi se référer à [Sethi and Thompson, 2006] pour des applications relatives aux problèmes économiques.

Dans le cas de contraintes du premier ordre sur l'état, [Bonnans and Hermant, 2008] a prouvé que l'état adjoint était continu sous certaines hypothèses. Pour des contraintes d'ordre supérieur cependant, des "conditions de saut" sur l'état adjoint doivent être considérées dans le problème d'optimisation, la continuité de celui-ci n'étant pas assurée.

L'inconvénient majeur de ces méthodes provient du fait qu'il soit nécessaire de connaître la forme de la trajectoire optimale pour pouvoir intégrer une équation différentielle valide, ce qui revient à connaître le nombre de contraintes actives.

3.3 Approche par l'équation de Hamilton-Jacobi-Bellman

3.3.1 Equation de Hamilton-Jacobi-Bellman et fonction valeur

Considérons à nouveau le problème d'optimisation général (3.1), qu'on englobe dans la classe des problèmes avec état et instants initiaux (x,t) quelconques, où x suit la loi d'évolution $\dot{x}(t) = f(x(t),u(t),t)$. On définit la fonction valeur (ou fonction coût) $V(x(t),t)$

$$V(x(t),t) := \min_{u \in U} \left[\int_t^T L(x(\tau),u(\tau),\tau) d\tau + \phi(x(T),T) \right]. \tag{3.7}$$

On suppose que $V(x(t),t)$ existe et est C^1. Soit Δt l'intervalle de temps, d'après l'équation d'évolution on a alors :

$$x(t+\Delta t) \approx x(t) + f(x(t),u(t),t)\Delta t. \tag{3.8}$$

Considérons qu'une commande quelconque soit appliquée durant le laps de temps Δt. Alors on aura au premier ordre :

$$V(x(t),t) \leq V(x(t)+f(x(t),u(t),t)\Delta t, t+\Delta t) + L(x(t),u(t),t)\Delta t. \tag{3.9}$$

L'égalité dans (3.9) n'est obtenue que si une commande optimale est appliquée durant l'intervalle de t à $t+\Delta t$. On peut donc écrire :

$$V(x(t),t) = \min_{u\in U}\{V(x(t)+f(x(t),u(t),t)\Delta t, t+\Delta t) + L(x(t),u(t),t)\Delta t\} \tag{3.10}$$

Par un développement de Taylor au premier ordre, on obtient :

$$V(x(t),t) = \min_{u\in U}\{V(x(t),t) + \frac{\partial V}{\partial x}(x(t),t)f(x(t),u(t),t)\Delta t + \frac{\partial V}{\partial t}(x(t),t)\Delta t + L(x(t),u(t),t)\Delta t\}. \tag{3.11}$$

Comme V et $\dfrac{\partial V}{\partial t}$ ne dépendent pas de u, on peut écrire, lorsque $\Delta t \to 0$

$$-\frac{\partial V}{\partial t}(x(t),t) = \min_{u\in U}\left[L(x(t),u(t),t) + \frac{\partial V}{\partial x}(x(t),t)f(x(t),u(t),t)\right]. \tag{3.12}$$

L'équation (3.12) est appelée équation de Hamilton-Jacobi-Bellman (ou équation HJB). Elle correspond à l'équation représentant le problème de minimisation (*PO*) (défini dans la section 2). La fonction valeur V est ainsi solution d'une équation aux dérivées partielles où apparaissent l'équation du système $\dot{x}(t) = f(x,u,t)$ et la fonction à minimiser $L(x,u,t)$. Trouver $V(x,t)$ dans (3.7) revient donc à résoudre l'équation (3.12).

3.3.2 Programmation Dynamique

La programmation dynamique est fréquemment utilisée pour résoudre les problèmes du type (3.1) (voir [Wu et al., 2002], [Scordia, 2004], [Sundström et al., 2008b] pour des applications sur véhicules hybrides) : cette méthode repose sur le principe d'optimalité de Bellman, énoncé par R. Bellman :

"Une suite de commandes optimales est telle que, quels que soient l'état et l'instant considérés sur une trajectoire optimale, les commandes ultérieures constituent pour le problème ayant cet état et cet instant comme éléments initiaux une suite de commandes optimales."

Le principe d'optimalité de Bellman indique qu'une commande optimale peut être construite séquentiellement, tout d'abord en calculant la commande optimale pour le dernier pas de temps, ensuite pour les deux derniers, puis pas à pas jusqu'à considérer le problème dans sa globalité.

Principe

La programmation dynamique est donc une méthode d'optimisation en temps rétrograde, qui permet de calculer une approximation de la fonction V solution de l'équation HJB, discrétisée en espace et en temps. Pour cela, on calcule de façon séquentielle la trajectoire optimale entre deux instants, pour un état initial donné.

On définit le Principe de Programmation Dynamique, valide $\forall x \in \mathbb{R}$ et $\forall t_k, t_{k+1}$ tels que $t_k < t_{k+1} \leq T$, avec $t_{k+1} = t_k + \Delta t$ par

$$V(x,t_k) = \min_{u\in U}\{\Delta t L(x,u,t_k) + V(x+\Delta t f(x,u,t_k), t_{k+1})\}. \tag{3.13}$$

et la fonction $V(x,t_k)$ s'écrit, au temps $t_k = T$

$$V(x,T) = \phi(x_N). \tag{3.14}$$

Justification de l'équation (3.13)

Pour tout (x_0,t_0) fixé et tout $u_0 \in U$ donné, on considère la trajectoire

$$t \mapsto \tilde{x}(t;x_0,t_0,u_0)$$

définie par le système différentiel

$$\begin{cases} \dfrac{d}{dt}\tilde{x}(t;x_0,t_0,u_0) = f(\tilde{x}(t;x_0,t_0,u_0),u_0,t_0) \\ \tilde{x}(t=t_0;x_0,t_0,u_0) = x_0 \end{cases} \tag{3.15}$$

dont la solution existe au voisinage de t_0. Soit la fonction composée

$$\tilde{V}(t;x_0,t_0,u_0) = V(\tilde{x}(t;x_0,t_0,u_0),t)$$

qui représente la valeur de V sur la trajectoire $\tilde{x}$. Alors, pour tout t, on a

$$\begin{aligned} \frac{d}{dt}\tilde{V}(t;x_0,t_0,u_0) &= \frac{\partial}{\partial t}V(\tilde{x}(t;x_0,t_0,u_0),t) + \frac{\partial}{\partial x}V(\tilde{x}(t;x_0,t_0,u_0),t)\frac{d}{dt}\tilde{x}(t;x_0,t_0,u_0) \\ &= \frac{\partial}{\partial t}V(\tilde{x}(t;x_0,t_0,u_0),t) + \frac{\partial}{\partial x}V(\tilde{x}(t;x_0,t_0,u_0),t)f(\tilde{x}(t;x_0,t_0,u_0),t_0,u_0). \end{aligned}$$

En spécifiant $t = t_0$, on obtient

$$\frac{d}{dt}\tilde{V}(t=t_0;x_0,t_0,u_0) = \frac{\partial}{\partial t}V(x_0,t_0) + \frac{\partial}{\partial x}V(x_0,t_0)f(x_0,t_0,u_0).$$

Cette relation est valide pour tout (x_0,t_0,u_0). La fonction composée $\tilde{V}$ nous aide à discrétiser l'équation HJB en temps. En effet, entre l'instant t_k et $t_{k+1} = t_k + \Delta t$, on a, pour tout $u \in U$

$$\frac{\partial}{\partial t}V(x,t_k) + f(x,u,t_k)\frac{\partial V}{\partial x}(x,t_k) = \frac{d}{dt}\tilde{V}(t=t_k;x,t_k,u),$$

qu'on peut approcher par la différence finie

$$\frac{\tilde{V}(t=t_{k+1};x,t_k,u) - \tilde{V}(t=t_k;x,t_k,u)}{\Delta t}.$$

Or, il est immédiat que

$$\tilde{V}(t=t_k;x,t_k,u) = V(x,t_k,u).$$

D'autre part

$$\tilde{V}(t=t_{k+1};x,t_k,u) = V(\tilde{x}(t=t_{k+1};x,t_k,u),t_{k+1}),$$

et on peut approcher la position $\tilde{x}(t=t_{k+1},x,t_k,u)$ par sa version discrète

$$\frac{\tilde{x}(t=t_{k+1};x,t_k,u) - x}{\Delta t} = f(x,u,t_k)$$

du système différentiel (3.15) pour $x_0 = x$, $t_0 = t_k$. Finalement, on a l'approximation

$$\frac{\partial V}{\partial t}(x,t_k) + f(x,u,t_k)\frac{\partial V}{\partial x}(x,t_k) \approx \frac{V(x+\Delta t f(x,u,t_k),t_{k+1}) - V(x+\Delta t f(x,u,t_k),t_{k+1})}{\Delta t}.$$

En reportant cette approximation dans l'équation HJB, et en multipliant par Δt, on aboutit à

$$\min_{u\in U}\{V(x+\Delta t f(x,u,t_k),t_{k+1})-V(x,t_k)+\Delta t L(x,u,t_k)\}=0.$$

Comme $V(x,t_k)$ ne dépend pas de u, on peut réécrire la relation précédente sous la forme d'un schéma rétrograde

$$V(x,t_k)=\min_{u\in U}\{V(x+\Delta t f(x,u,t_k),t_{k+1})+\Delta t L(x,u,t_k)\}. \tag{3.16}$$

On retrouve ainsi le principe de Programmation Dynamique.

3.3.3 Méthode de résolution

Pour l'application de la programmation dynamique, on doit résoudre l'équation (3.13) à chaque pas de temps. On se place maintenant dans l'ensemble $[x_{\min},x_{\max}]\subset\mathbb{R}$. Il n'est cependant pas possible de résoudre cette équation $\forall x\in[x_{\min},x_{\max}]$. Aussi, on a recours à une discrétisation de l'espace d'état, l'équation (3.13) étant alors évaluée à chaque pas de temps, et à chaque pas d'espace. Cette discrétisation dans le cas d'un problème à un seul état impose de parcourir une grille, l'axe horizontal représentant le temps, l'axe vertical l'espace. Pour chaque point de la grille, un certain nombre de commandes admissibles sont testées, la commande optimale retenue étant celle qui minimise le terme de droite dans (3.13).

L'algorithme de programmation dynamique s'écrit :

Algorithme 1: Programmation Dynamique

Données: $x_{\min}$, $x_{\max}$, Δx, Δt, x_0, T
Définir $V(x,T)=\beta(x_0-x)^2$
début
  pour $t_k=(T-\Delta t):-\Delta t:0$ **faire**
    pour $x=x_{\min}:\Delta x:x_{\max}$ **faire**
      Calculer $u^*(x,t_k)=\underset{u\in U}{\operatorname{argmin}}\{V(x+\Delta t f(x,u,t_k),t_{k+1})+\Delta t L(x,u,t_k)\}$
    fin
  fin
  $x(0)=x_0$;
  pour $t_k=0:\Delta t:(T-\Delta t)$ **faire**
    $x(t_{k+1})=f(x(t_k),u^*(x(t_k),t_k),t_k)$
  fin
fin

Cet algorithme comporte donc une partie *backward*, qui résout le problème d'optimisation défini par (3.13) un certain nombre de fois (voir section 3.3.4) du temps $t=T-1$ au temps $t=0$. Cette étape terminée, l'ensemble des commandes optimales sont connues pour chaque noeud de la grille de discrétisation. Il suffit ensuite de parcourir la grille dans le sens *forward*, de l'état au temps initial $x(0)=x_0$ jusqu'au temps final $x(T)$, en sélectionnant la commande optimale du noeud près duquel on se trouve.

Le calcul de $V(x+\Delta t f(x,u,t_k),t_{k+1})$ nécessite l'utilisation d'un schéma numérique. Nous choisissons un schéma explicite, tel qu'il est décrit dans [Guilbaud, 2002] : le *schéma décentré* ou *schéma upwind*.

Il est défini de la façon suivante :

$$\begin{cases} V(x^i,t_k) & = \min\limits_{u\in U}\Big[\Delta t L(u,t_k) + V(x^i,t_{k+1}) + \max(f(x^i,u,t_k),0)\dfrac{V(x^i,t_{k+1}) - V(x^{i-1},t_{k+1})}{\Delta x}\Delta t \\ & \quad + \min(f(x^i,u,t_k),0)\dfrac{V(x^{i+1},t_{k+1}) - V(x^i,t_{k+1})}{\Delta x}\Delta t\Big] \\ V(x^i,T) & = \phi(x^i,T), \end{cases} \tag{3.17}$$

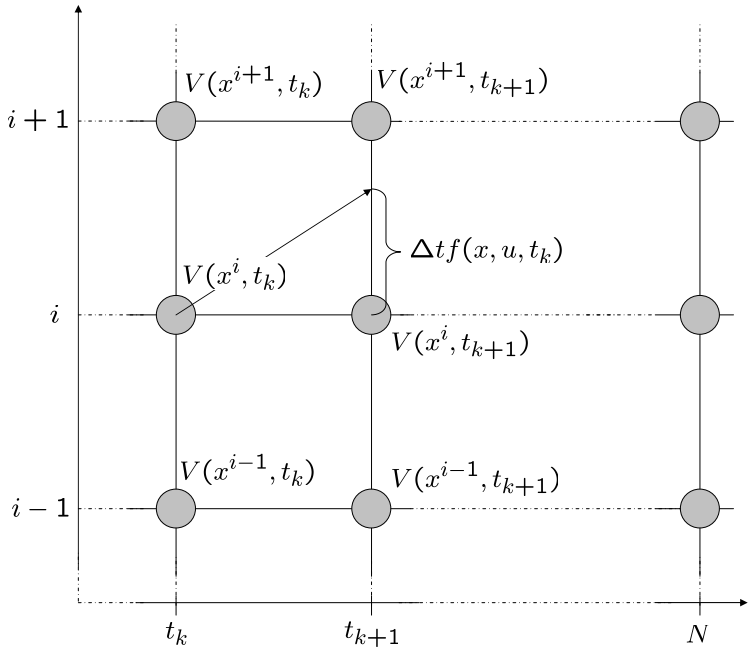

FIG. 3.1: *Schéma upwind pour le calcul de* $V(x+\Delta t f(x,u,t_k),t_{k+1})$

Pour l'utilisation de ce schéma, et pour en garantir la stabilité, la condition CFL (Courant, Friedrichs, Lewy) doit être vérifiée. Cette condition nous assure qu'en tout point de la grille $x^i(t_k)$, et pour tout $u \in U$, le point

$$x^i(t_k) + \Delta t f(x^i,u,t_k)$$

se trouve dans l'enveloppe convexe des voisins immédiats de x^i. Elle s'exprime sous la forme

$$\frac{\Delta t}{\Delta x}|f| \leq 1$$

Contrairement à [Guilbaud, 2002] qui propose d'utiliser la méthode dite du "trapèze", c'est à dire une grille agrandie qui se réduit d'un cran dans les directions d'espace à chaque pas de temps, on choisi de conserver une grille de dimension constante. Afin de ne pas avoir à introduire de condition artificielle en dehors du domaine qui consistait à pondérer les valeurs calculées lorsque les contraintes d'état sont franchies, nous choisissons de calculer préalablement les contrôles admissibles, ceux-ci tenant compte des contraintes sur l'état. Ainsi, si l'on se trouve près d'un bord, seuls les contrôles admissibles permettant de ne pas franchir la contrainte sur l'état seront retenus.

La valeur de la fonction $V(x,t)$ à $t = T$ est choisie égale à $\beta(x_0 - x)^2$, permettant d'imposer un coût à $V(x,T)$ dès que x s'écarte trop de x_0. Le coefficient β est choisi arbitrairement.

Pour gérer l'interpolation de la fonction coût dans le cas de forts gradients, on pourra aussi se référer à [Sundström et al., 2008a].

3.3.4 Commentaires sur la méthode de programmation dynamique

Le principal désavantage de la programmation dynamique est son temps de calcul. En effet, même si le problème d'optimisation à résoudre, grâce à la résolution de l'équation HJB discrétisée, est très simple, il doit être effectué un très grand nombre de fois.

Pour Δt et Δx les pas de temps et d'espace, on peut définir les nombres de pas de temps N_t et d'espace N_x nécessaires à la construction de la grille. Ils sont donnés par

$$N_t = T/\Delta t \tag{3.18}$$
$$N_x = (x_{\max} - x_{\min})/\Delta x. \tag{3.19}$$

Pour minimiser le terme $V(x + \Delta t f(x,u,t_k), t_{k+1}) + \Delta t L(x,u,t_k)$ dans l'expression (3.13), on teste un nombre fini de valeurs. On note N_u ce nombre. En notant Δt_{mod} le temps nécessaire à l'évaluation de $L_k(x_k,u_k)$ et de $f_k(x_k,u_k)$, le temps d'exécution global est donné par la formule

$$T_{glb} = N_t N_x N_u \Delta t_{mod}, \tag{3.20}$$

où le produit $N_t N_x N_u$ représente le nombre d'évaluations du modèle, pour un triplet (x,u,t) donné. Typiquement, pour le type de problème lié à notre application, on choisit des pas de temps, d'espace, et un nombre N_u égaux à $N_t \approx 2000$, $N_x \approx 100$, $N_u = 50$, ce qui conduit à 10.000.000 évaluations du modèle. Il est donc impératif d'utiliser un modèle simple pour l'évaluation de $L_k(x_k,u_k)$ et $f_k(x_k,u_k)$, sans quoi le temps de calcul devient prohibitif.

On notera que si l'on considère non plus une seule, mais deux variables d'état x_1 et x_2, le nombre d'évaluations s'écrira $N_t N_{x_1} N_{x_2} N_u$, ce qui augmente considérablement le temps de calcul. C'est pourquoi la programmation dynamique est bien adaptée à des problèmes dont le nombre de variables d'état est faible.

La commande optimale est obtenue sous la forme d'une matrice de dimension N_t, N_x, celle-ci étant tabulée suivant la discrétisation appliquée lors de la résolution. Durant le parcours *forward*, la commande optimale au pas k doit être choisie en recherchant la paire (x_k,t_k) (de la grille discrétisée) la plus proche du point courant, puis en sélectionnant la commande optimale qui avait été sauvegardée pour cette paire.

3.3.5 Résolution du problème (1.2) avec la programmation dynamique

Dans le problème continu (1.2), r peut prendre les valeurs 0 ou 1, sa dérivée $\dot{r}$ n'étant pas bornée. Pour résoudre ce problème avec la programmation dynamique, on conserve cette seconde variable d'état discrète, mais on suppose aussi que sa dérivée $\dot{r} = v$ est discrète.

Les équations d'évolution des deux variables d'état x et r, une fois discrétisées, nous permettent d'écrire

$$\begin{cases} x(t_{k+1}) = x(t_k) + f(t_k, u(t_k), r(t_k))\Delta t \\ r(t_{k+1}) = r(t_k) + v(t_k)\Delta t, \end{cases} \tag{3.21}$$

où Δt correspond au pas de discrétisation utilisé dans la programmation dynamique. Ici, on suppose que v est bornée, et peut prendre les valeurs $\{0, -\frac{1}{\Delta t}, \frac{1}{\Delta t}\}$. Cette paramétrisation nous assure que le terme $v\Delta t$ ne prend que des valeurs entières, ce qui constitue une condition nécessaire pour que r ne puisse prendre que les valeurs $\{0,1\}$. Le coût de redémarrage (voir section (1.3)) s'écrit donc $C_0 \max(0,v)\Delta t$, puisque ce coût doit rester indépendant du pas de discrétisation choisi.

Le problème discrétisé qui est résolu par la programmation dynamique s'écrit alors

$$\begin{cases} \min\limits_{u(t_k)\in U_k, v(t_k)\in\{0,-\frac{1}{\Delta t},\frac{1}{\Delta t}\}} \sum\limits_{k=0}^{N-1} (r(t_k)L(t_k,u(t_k))\Delta t + C_0 \max(0,v(t_k))\Delta t) + \phi(x_N) \\ \text{avec :} \\ x(t_{k+1}) = x(t_k) + f(t_k,u(t_k),r(t_k))\Delta t, \quad x(0) = x_0 \\ r(t_{k+1}) = r(t_k) + v(t_k)\Delta t \\ x_{\min} \leq x_k \leq x_{\max} \\ r \in \{0,1\}, \end{cases} \tag{3.22}$$

où $U_k = \{u(t_k) \mid u_{\min} \leq u(t_k) \leq u_{\max}\}$ est l'ensemble des commandes admissibles au temps t_k. La fonction valeur associée à ce problème à deux variables d'état x et r s'écrit finalement

$$V_k(x,r) = \min_{u\in U_k, v\in\{0,-\frac{1}{\Delta t},\frac{1}{\Delta t}\}} \{rL(x,u,t_k)\Delta t + C_0 \max(0,v)\Delta t + V(x+\Delta t f(u,r,t_k), r+v\Delta t, t_{k+1})\}. \tag{3.23}$$

La Figure 3.2 montre la grille de discrétisation utilisée pour la résolution de l'équation (3.23). On remarquera que dans le cas où $r=0$ aux temps t_k et t_{k+1}, i.e. $v(t_k)=0$, le mode de fonctionnement obligatoire est le mode électrique pur, ce qui implique que la variation $\Delta t f(r,u,t_k)$ soit unique et indépendante de u.

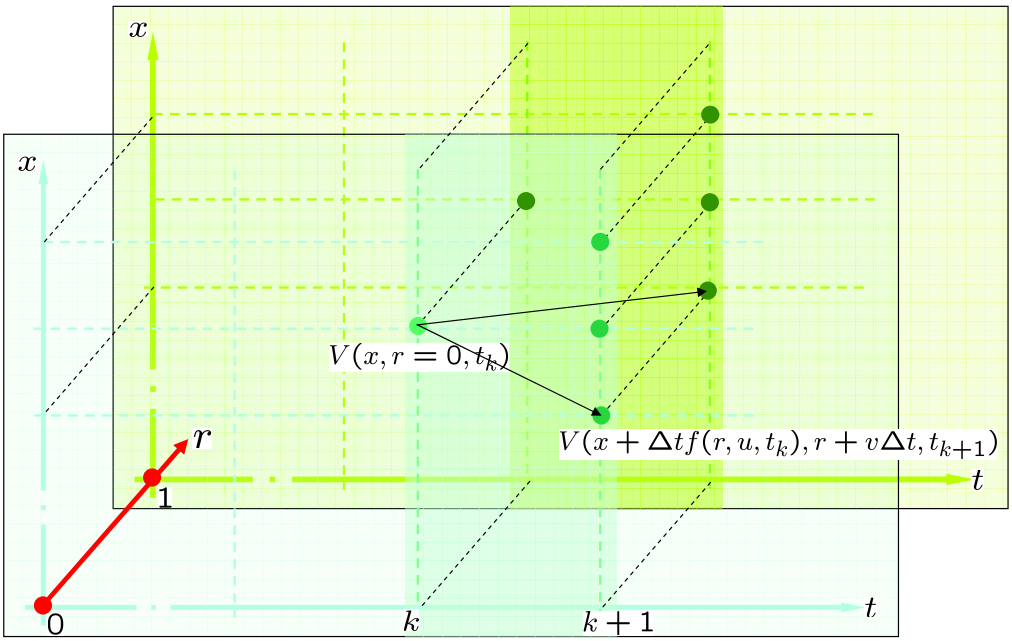

FIG. 3.2: *Grille de discrétisation utilisée en programmation dynamique pour la résolution du problème (1.2).*

3.4 L'algorithme SCOP

L'algorithme SCOP (Sequential Constraint-free Optimal control Problem algorithm) est une méthode originale de résolution séquentielle utilisant une méthode de tir simple. Elle permet de résoudre des problèmes qui sont exactement du type (1.1), que l'on rappelle ici :

$$(PO_2)\begin{cases} \min\limits_{u}\left\{J(u)=\displaystyle\int_0^T L(x(t),u(t),t)dt\right\} \\ \text{avec :} \\ \dot{x}=f(x(t),u(t),t), \quad x(0)=x_0, \quad x(T)=x_T, \\ x_{\min}\leq x(t)\leq x_{\max} \\ u_{\min}(t)\leq u(t)\leq u_{\max}(t), \end{cases} \tag{3.24}$$

c'est-à-dire des problèmes d'optimisation de systèmes dynamiques avec contraintes de bornes sur l'état, et dont l'état initial $x(0)$ et l'état final $x(T)$ sont imposés.

3.4.1 Principe

Le principe de SCOP repose sur une résolution séquentielle d'un problème d'optimisation utilisant un algorithme de tir simple. Contrairement aux algorithmes de tir multiple, pour lesquels la trajectoire doit être connue préalablement, et qui nécessitent une valeur adéquate pour l'initialisation de la valeur du multiplicateur associé à la dynamique sur l'état (voir section (3.2)), SCOP peut fonctionner sans connaissance préalable de la trajectoire optimale, ou des contraintes sur l'état qui deviendront actives.

3.4.2 Description de la méthode

L'idée de l'algorithme consiste à résoudre le problème (3.24) en résolvant séquentiellement le sous-problème sans contrainte d'état $(\mathscr{Q}_k)$ défini par

$$(\mathscr{Q}_k)[t_k,X_k] \qquad \min_{u(\cdot)\in U}\int_{t_i}^{t_k} L(x_k(t),u(t),t)\,dt \tag{3.25}$$

avec

$$\frac{dx_k}{dt}(t)=f(x_k(t),u(t),t) \tag{3.26a}$$
$$x_k(t_i)=\zeta \tag{3.26b}$$
$$x_k(t_k)=X_k, \tag{3.26c}$$

où $x_k(t_i)$ correspond à une valeur initiale en t_i, et (t_k,x_k) sont deux paramètres pouvant évoluer jusqu'à ce que $x(t_k)$ atteigne une valeur "correcte".
L'algorithme complet s'écrit

Algorithme 2: SCOP

données $k = 0$, $t_i = 0$, $t_0 = T$, and $X_0 = \zeta$;
début
  Résoudre $(\mathscr{Q}_0)[t_0, X_0]$
  tant que $t_i \neq T$ **faire**
    tant que $\exists t \in]t_i, t_k[\mid x_k(t) \leq x_{\min}$ **ou** $x_k(t) \geq x_{\max}$ **faire**
      Définir $\Delta_k(t) = \max\{x_{\min} - x_k(t), x_k(t) - x_{\max}\}$
      Determiner $t_{k+1} = \text{argmax}_{t \in [t_i, t_k]} \Delta_k(t)$
      Calculer $X_{k+1} = \Pi_{[x_{\min}, x_{\max}]}(x_k(t_{k+1}))$
      Résoudre $(\mathscr{Q}_{k+1})[t_{k+1}, X_{k+1}]$
      $k = k + 1$;
    fin
    si $t_{k+1} == T$ **alors**
      Arrêt de l'algorithme, trajectoire optimale obtenue.
      $t_i = T$
    sinon
      $t_i = t_{k+1}$; $\zeta = X_{k+1}$; $t_{k+1} = T$
    fin
  fin
fin

Le symbole $\Pi_{[x_{\min}, x_{\max}]}(X)$ correspond à l'opérateur de troncature

$$\Pi_{[x_{\min}, x_{\max}]}(X) = \begin{cases} x_{\min} & \text{si } X < x_{\min} \\ x_{\max} & \text{si } X > x_{\max}. \end{cases} \tag{3.27}$$

Dans cet algorithme, on cherche à résoudre le problème $(\mathscr{Q}_k)$ de manière séquentielle, jusqu'à obtenir une trajectoire optimale. L'intervalle de temps pour lequel l'optimisation est effectuée est restreint à chaque fois qu'une contrainte d'état est activée. Le calcul de $x(t)$, $t \in [0, T]$ est réalisé en utilisant un schéma explicite classique, la trajectoire étant alors discrétisée en temps.

Lorsque l'état x longe une contrainte de borne, la résolution du problème $(\mathscr{Q}_{k+1})[t_{k+1}, X_{k+1}]$, pour le temps initial t_i, mène à $t_{k+1} = t_{i+1}$, le problème (3.25, 3.26) étant alors résolu entre deux pas de temps (t_i, t_{i+1}). C'est notamment le cas du problème qui est présenté dans l'article, section (3.4.6), dans lequel il s'agit de minimiser l'énergie d'une corde reposant sur un support plan.

3.4.3 Exemple sur un cas simple

On considère le problème d'optimisation suivant :

$$(PO3) \begin{cases} \min\limits_{u} \left\{ J(u) = \displaystyle\int_0^T \alpha^2(t) u^2(t) dt \right\} \\ \text{avec :} \\ \dot{x}(t) = (u(t) - 1)\alpha(t), \quad x(0) = 0,5, \quad x(T) = 0,5, \end{cases} \tag{3.28}$$

Ce problème sans contrainte se résout aisément. En écrivant l'Hamiltonien

$$H(t, u, p) = \alpha^2(t) u^2(t) + p(t)(u(t) - 1)\alpha(t), \tag{3.29}$$

les conditions d'optimalité mènent à

$$\begin{aligned} \dot{p} &= 0 \\ \frac{\partial H}{\partial u} &= 0, \end{aligned}$$

soit

$$\begin{aligned} p &= p_0 \\ u^* &= -\frac{p}{2\alpha}. \end{aligned}$$

La commande optimale dépend donc du scalaire p_0, qui devient la nouvelle inconnue. Il suffit d'intégrer l'équation du système, en utilisant l'expression de $\alpha(t)$ pour trouver la valeur de p_0 :

$$\int_0^T \dot{x}(t)dt = \int_0^T (u^* - 1)\alpha(t)dt.$$

On choisit $\alpha(t)$ strictement croissant sur [0,T], tel que H soit fortement convexe. Avec $\alpha(t) = (t^2 + 200t + 5000)/10000$, et $T = 20$, on trouve $p_0 = -1,4267$.

La commande optimale s'écrit donc : $u^*(t) = \dfrac{1,4267}{2\alpha(t)}$. La Figure 3.3 montre les résultats obtenus.

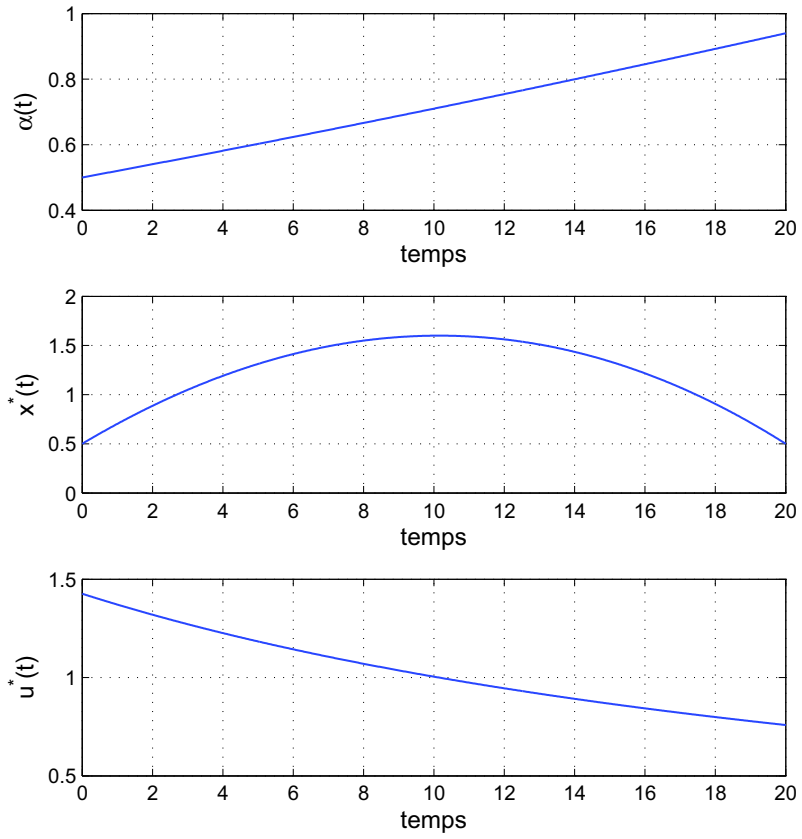

FIG. 3.3: *Trajectoires optimales pour le problème (3.28) sans contrainte sur l'état :* $\alpha(t)$ *(haut),* $x^*(t)$ *(milieu) et* $u^*(t)$ *(bas).*

On introduit maintenant une contrainte de borne sur x : $x \leq 1$. Il est clair que le contrôle précédemment trouvé ne permet pas de satisfaire cette contrainte, la trajectoire passant au delà de $x = 1,5$. Ce nouveau problème peut être résolu avec diverses méthodes : la programmation dynamique, une méthode de Tir multiple (grâce à la connaissance de la forme de la trajectoire), etc. Nous choisissons ici SCOP, pour sa rapidité d'exécution et sa simplicité. L'application de SCOP sur ce nouveau problème nous permet d'obtenir les trajectoires suivantes, Figure 3.4 :

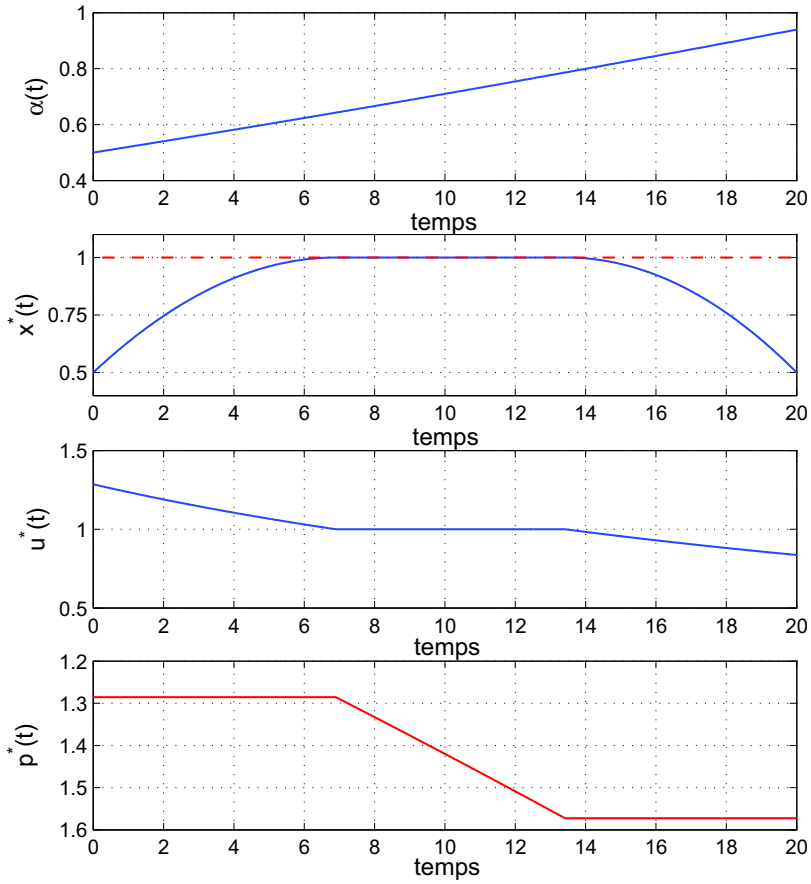

FIG. 3.4: *Trajectoires optimales pour le problème (3.28), contraint par $x \leq 1$: $\alpha(t)$ (haut), $x^*(t)$ (haut), $u^*(t)$ (bas) et $p^*(t)$ (bas).*

On constate que la trajectoire optimale de l'état $x^*(t)$ vérifie bien la contrainte imposée $x \leq 1$. Le multiplicateur $p^*(t)$ n'est plus constant sur [0 20] : il reste constant lorsque les contraintes d'état sont inactives, mais évolue ensuite dès qu'une contrainte est saturée. On peut retrouver numériquement son évolution, étant donné que sur la portion où $x = 1$, on a nécessairement $u = 1$, ce qui impose $p = -2\alpha$. La trajectoire optimale de $x(t)$ a été validée en résolvant le même problème d'optimisation avec la programmation dynamique.

On suppose finalement qu'une légère perturbation s'applique sur $\alpha(t)$ lorsque la contrainte $x \leq 1$ est d'ordinaire active. Cette perturbation, si elle est relativement peu élevée, peut se trouver "effacée" par les erreurs de discrétisation lors de la résolution par un algorithme classique, comme la programmation dynamique (voir résultats Figure 3.6). SCOP nous permet d'obtenir les résultats suivants :

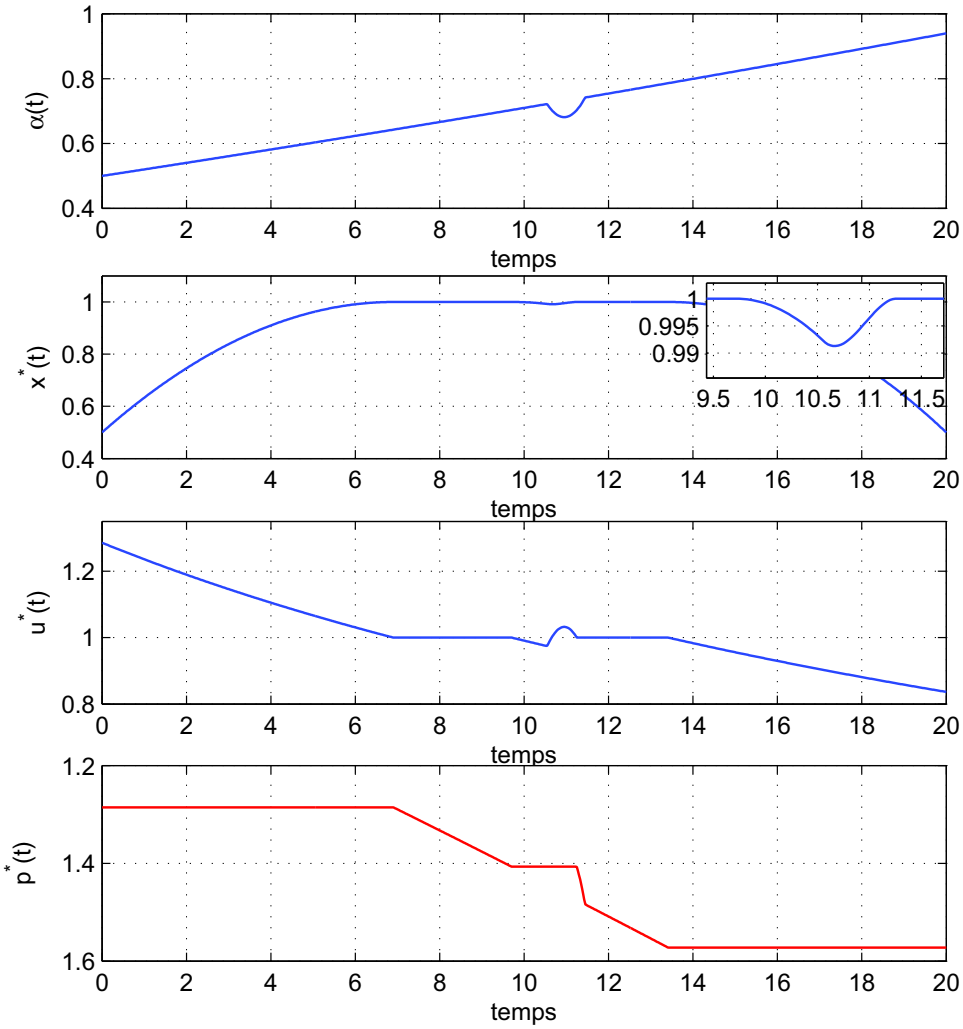

FIG. 3.5: *Trajectoires optimales pour le problème (3.28), contraint par* $x \leq 1$ *et* α *modifié.*

Il est notamment intéressant de comparer le temps de calcul nécessaire à une résolution par programmation dynamique, à celui que l'on obtient avec SCOP. La Figure 3.6 indique le temps nécessaire à l'obtention des différentes courbes.

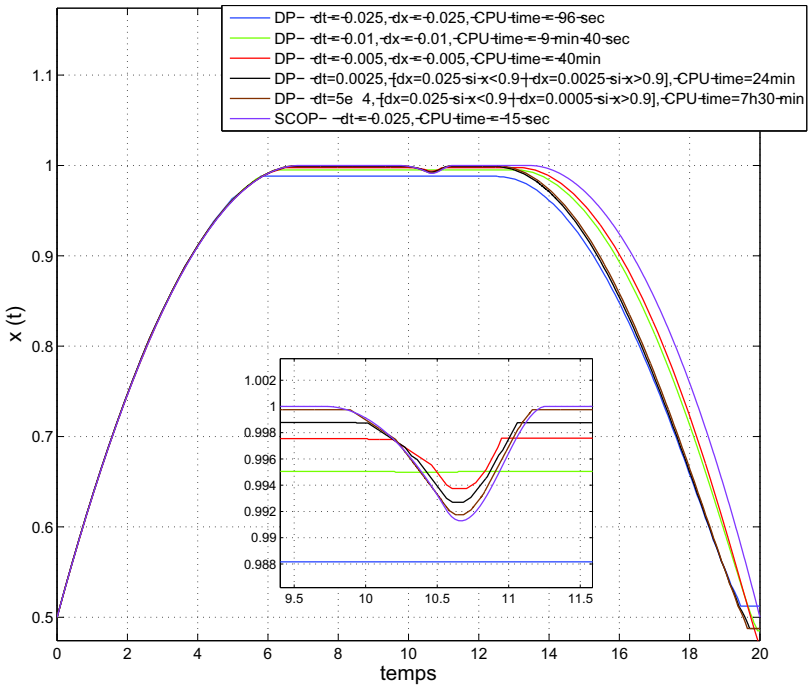

FIG. 3.6: *Trajectoires optimales & temps de calcul avec SCOP et Programmation Dynamique (DP) pour le cas présenté Figure 3.5.*

Le temps nécessaire pour la convergence de SCOP est d'environ 15 secondes, tandis que pour obtenir une courbe proche de celle de SCOP, la programmation dynamique requiert plus de 7 heures, qui plus est en utilisant un maillage plus fin à proximité de la contrainte d'état $x \leq 1$. On remarque que sur certaines courbes, la contrainte sur la valeur de l'état final est mal vérifiée, cela peut être partiellement corrigé en pénalisant davantage cette contrainte dans l'expression du coût au temps T.

3.4.4 Commentaires sur la méthode SCOP

L'algorithme SCOP a été appliqué sur différents problèmes, avec les hypothèses suivantes :

- une ou deux contraintes de bornes sur un seul état, indépendantes de la variable d'évolution (le temps t, ou bien l'abscisse x dans le cas d'une corde suspendue),
- une équation du système linéaire en x (ou sans dépendance en x).

Les limites de cet algorithme concernent le domaine d'applicabilité, qui est encore mal connu. Cet algorithme a été appliqué à des problèmes dont l'équation du système est telle que $\dot{x}$ dépend linéairement de x, voire ne contient pas x dans l'équation d'évolution. La convergence de SCOP vers la solution optimale a été démontrée pour deux applications différentes (voir section 3.4.6). En dehors de ces hypothèses, la convergence de l'algorithme n'a pas été testée.

3.4.5 Application aux véhicules hybrides

Le problème de l'optimisation de la répartition de couple sur véhicule hybride est peu différent de celui traité dans la section 3.4.3. Le terme $\alpha(t)$ est plus complexe, et correspond à une demande de couple, et le terme à minimiser, noté L est une fonction convexe en couple.

Hypothèses de simplification

Il n'est a priori pas possible d'appliquer l'algorithme SCOP sur des problèmes d'optimisation du type (1.2), à cause du terme $C_0 \max(0, \dot{r})$, le critère à minimiser dépendant alors de la variable d'état discrète r et de sa dérivée. On considérera donc ici le problème du type (1.1).

D'autre part, afin de disposer d'une solution analytique, on approche la cartographie de consommation par un polynôme du second degré, noté L_p, polynôme convexe en couple T_e. Cette approximation nous permet de modéliser simplement la consommation de carburant avec une faible erreur (par rapport à la cartographie réelle).

Enfin, on considère que le rendement du moteur électrique est constant. Le problème d'optimisation à résoudre est donc le problème (3.24) avec $L = L_p$.

Résolution

Pour résoudre le problème d'optimisation, on passe par une série de sous-problèmes, de la forme définie par (3.25) et (3.26). Ces sous-problèmes sont résolus par une méthode de tir : on forme l'Hamiltonien, et l'on minimise celui-ci sur l'espace propre au sous-problème, sous la contrainte d'atteindre un certain état final. L'Hamiltonien s'écrit

$$H(u,t,p) = L_p(u,t) + p(t) f(u,t). \tag{3.30}$$

Comme H ne dépend pas de x, la condition d'optimalité (2.3b) mène à $\dot{p} = 0$, c'est à dire $p = p_0$. En omettant les contraintes sur la commande, on trouve le minimum de H en calculant $\dfrac{\partial H}{\partial u} = 0$, ce qui conduit à

$$\frac{\partial L_p}{\partial u} + p_0 \frac{\partial f}{\partial u} = 0. \tag{3.31}$$

Avec $L(\omega_e, T_e) = \sum_{i,j=0}^{2} K_{ij} \omega_e^i T_e^j$, la commande optimale u_{nc}^* du problème non contraint s'écrit

$$u_{nc}^*(t) = -\frac{\sum_{i=0}^{2} K_{i1} \omega(t)^i + p_0 K \omega(t)}{2 \sum_{i=0}^{2} K_{i2} \omega(t)^i T_{rq}(t)}. \tag{3.32}$$

Il suffit ensuite de projeter $u_{nc}^*(t)$ sur l'espace admissible $[u_{\min}(t), u_{\max}(t)]$ pour connaître la commande optimale $u^*(t)$.

Comparaison résultats de SCOP / Programmation dynamique

SCOP est appliqué sur le cas du véhicule étudié dans le Chapitre 4, et comparé aux résultats obtenus avec la programmation dynamique. On choisit volontairement le cycle Artemis sur lequel doit être optimisée la répartition de couple, celui-ci n'étant pas construit artificiellement comme le cycle NEDC, et comportant davantage de variété dans les conditions de fonctionnement des deux moteurs électrique et thermique. La taille de la batterie a été volontairement considérée égale à 150kW/h, pour que la trajectoire d'état de charge soit contrainte par les bornes admissibles, $x_{\min} = 50\%$ et $x_{\max} = 70\%$.

La Figure 3.7 représente les trajectoires obtenues par SCOP et par la programmation dynamique, pour le même suivi de cycle.

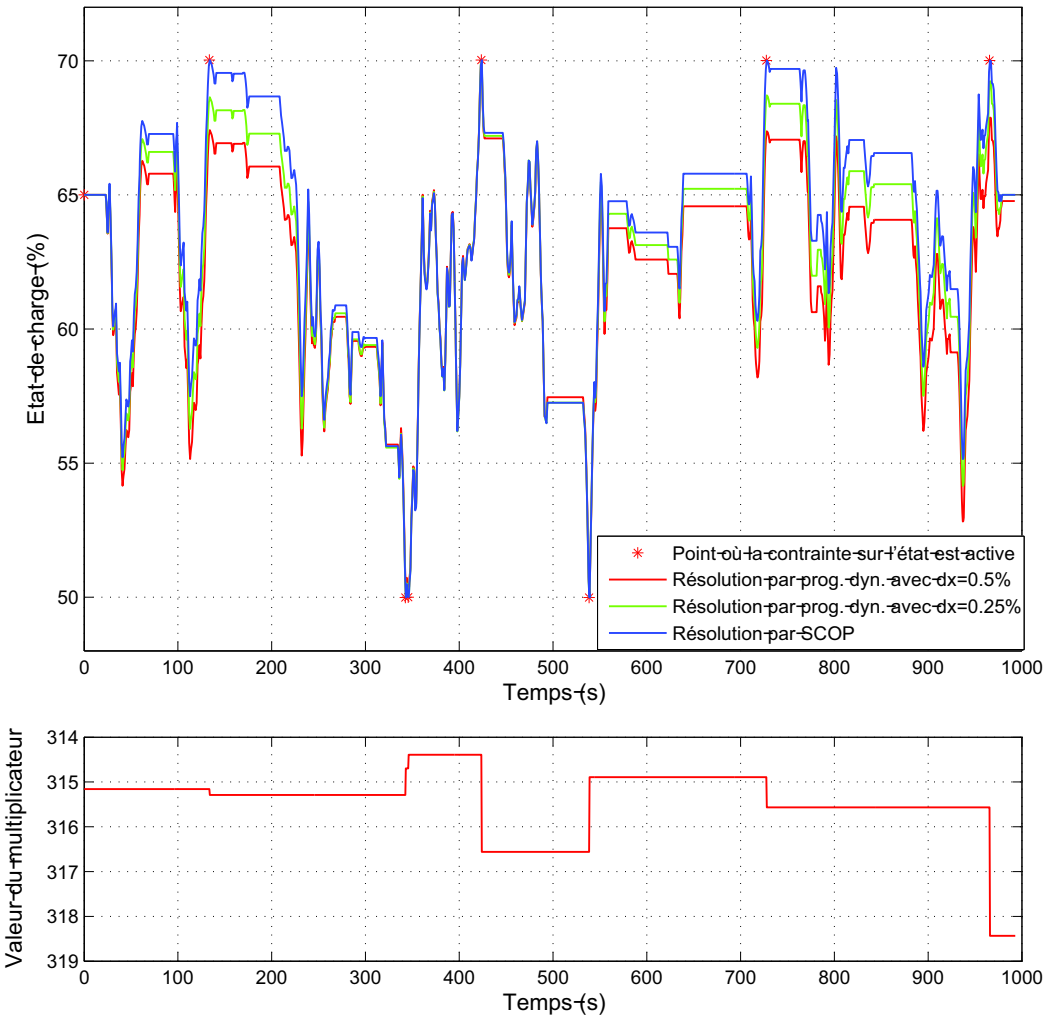

FIG. 3.7: *Comparaison de trajectoires optimales obtenues par programmation dynamique et SCOP (haut), trajectoire optimale du mutiplicateur de Lagrange p à l'aide de SCOP (bas)*

On peut remarquer que les trajectoires optimales de l'état obtenues par la programmation dynamique tendent à se rapprocher de la solution donnée par SCOP à mesure que l'on réduit le pas de discrétisation sur l'état. Des courbes similaires comparant la programmation dynamique à SCOP, pour une autre application de véhicule hybride, se trouvent dans [Rousseau et al., 2006b].

L'algorithme SCOP permet de connaître les valeurs du multiplicateur $p(t)$ associé à la trajectoire optimale de l'état (voir Figure 3.7 (bas)). Ainsi, la connaissance de ces valeurs est particulièrement intéressante pour la mise au point de lois temps-réels, telles que l'ECMS (voir section 5.1.3).

Enfin, grâce à la connaissance de $p(t)$, on peut tracer les trajectoires $(\omega_e(t), T_e(t))$, en résolvant l'équation $\dfrac{\partial H}{\partial u} = 0$, et en exprimant le couple T_e en fonction du régime ω_e. La Figure 3.8 représente ces courbes, ainsi que les points de fonctionnement du moteur thermique (ω_e, T_e), du moteur électrique, et les points de consigne, l'ensemble étant exprimé au niveau du moteur thermique.

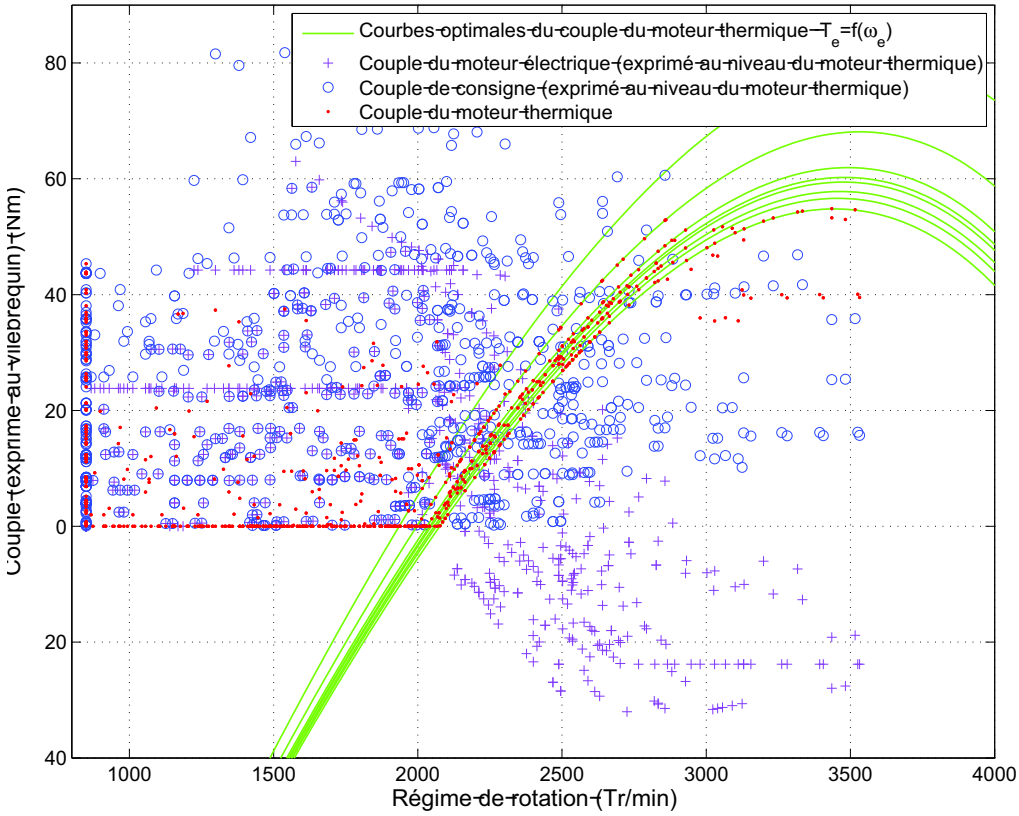

FIG. 3.8: *Couple de consigne, couple du moteur électrique (ramené au niveau du moteur thermique) et couple du moteur thermique en fonction du régime. / Faisceaux de courbes représentant les points de fonctionnement optimaux du moteur thermique pour le véhicule de référence sur le cycle Artemis.*

On remarque aisément que la majorité des points de fonctionnement se situe sur les courbes vertes. Lorsque c'est possible, le moteur thermique se place sur une de ces courbes, utilisant soit le mode régénération pour fournir un couple T_e supérieur au couple de consigne, soit le mode boost pour fournir un couple T_e inférieur au couple de consigne. Comme l'expression $\frac{\partial H}{\partial u} = 0$, dont sont dérivées les courbes optimales du couple du moteur thermique, ne prennent pas en compte les contraintes sur u, les points de fonctionnement du moteur thermique ne peuvent tous se placer sur ces courbes.

De plus, en traçant les points de consigne (qui correspondent aussi aux points de fonctionnement du moteur thermique pour un véhicule identique non muni d'un moteur électrique), ainsi que les points de fonctionnement optimaux du moteur thermique pour le véhicule hybride, que l'on superpose à la cartographie de consommation spécifique (issue du calcul polynomial, donc légèrement différente de la Figure 4.1 qui représente la vraie cartographie), Figure 3.9, on remarque que les points de fonctionnement optimaux ne sont pas forcément ceux qui sont placés dans la zone de meilleur rendement. C'est en fait un compromis global entre la minimisation de la consommation, la contrainte d'égalité sur l'état final, et la prise en compte de l'ensemble des contraintes sur la commande.

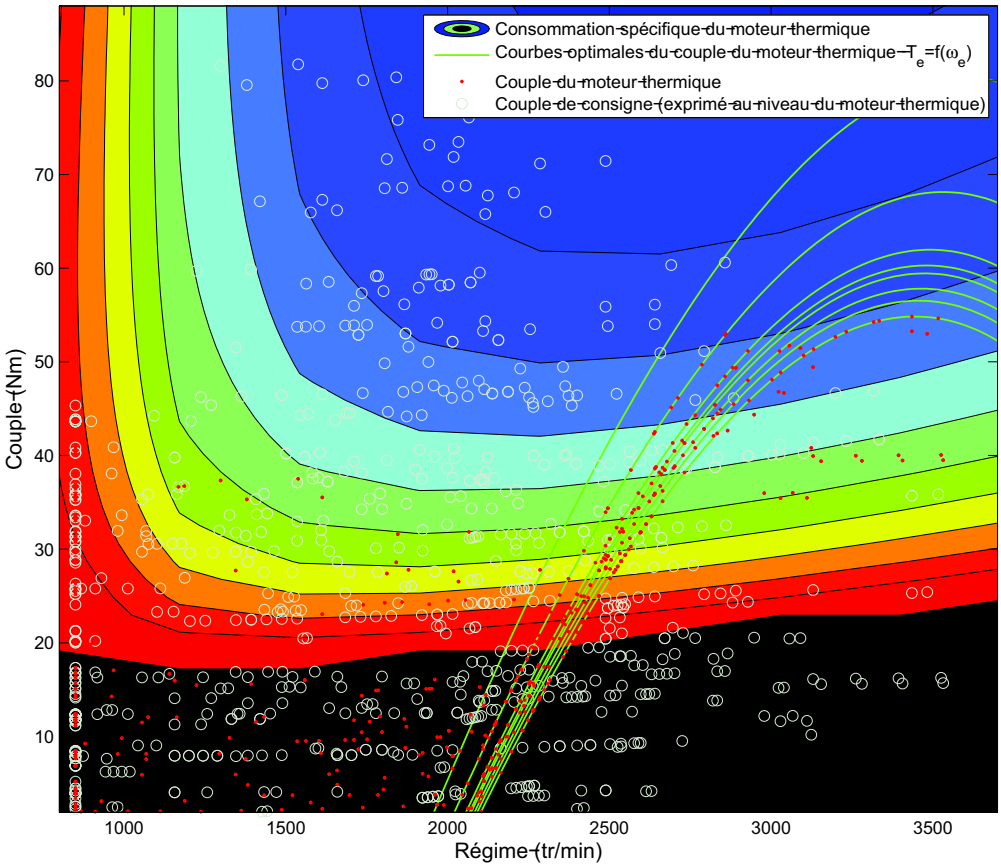

FIG. 3.9: *Couple de consigne, couple du moteur électrique (ramené au niveau du moteur thermique) et couple du moteur thermique en fonction du régime. / Faisceaux de courbes représentant les points de fonctionnement optimaux du moteur thermique pour le véhicule de référence sur le cycle Artemis.*

3.4.6 Etude de convergence de SCOP

La démonstration de la convergence de SCOP est présentée dans l'article nommé *SCOP : a Sequential Constraint-free Optimal control Problem algorithm* qui fut présenté au CCDC 2008 (Chinese Control and Decision Conference), à Yantaï, Chine (voir Annexes A).

Cette démonstration a été menée pour deux cas simples :

1. L'optimisation de la répartition de couple sur un modèle très simplifié de véhicule hybride, dans le cas d'une demande de puissance monotone sur [0,T].
2. La minimisation énergétique d'une corde reposant sur un support plan, et attachée à ses deux extrémités,

3.4.7 Perspectives

Les perspectives de SCOP sont assez larges, d'un point de vue du champs d'application. Cet algorithme a été testé dans le cas d'un système à un seul état, sur deux cas d'application assez différents. Sa

convergence a été démontrée sur ces deux applications, l'une d'elle (le véhicule hybride) sous certaines hypothèses, notamment de monotonie sur $\alpha(t)$. Néanmoins on constate, par comparaison avec la programmation dynamique, que SCOP fonctionne aussi sans l'hypothèse de monotonie sur $\alpha(t)$, et avec la présence de contraintes de borne dépendant du temps sur $u(t)$.

SCOP a aussi été testé avec une vraie cartographie de consommation (non polynomiale), pour laquelle il faut procéder à un vrai "tir", en procédant par une minimisation sur l'Hamiltonien H à chaque pas de temps. Dans ce cas, il n'existe pas de solution analytique pour u, le temps de convergence de SCOP augmente donc de manière substantielle.

L'utilisation de SCOP pour la résolution de problèmes de contrôle optimal dont l'état ne serait plus un scalaire n'a pas été étudiée, ni son implémentation. Pour un système comportant plusieurs états, la résolution devra notamment gérer les situations où plusieurs états violent des contraintes de façon simultanée.

D'autre part, la validité des résultats de SCOP dans le cas d'un problème dont les contraintes d'état dépendraient du temps est encore inconnue. Il serait néanmoins intéressant de poursuivre les recherches dans ce sens.