Informatique & Mathématiques Appliquées

TP-Projet Contrôle optimal, IMA 2A

Olivier Cots & Jérémy Rouot

22 mai 2017

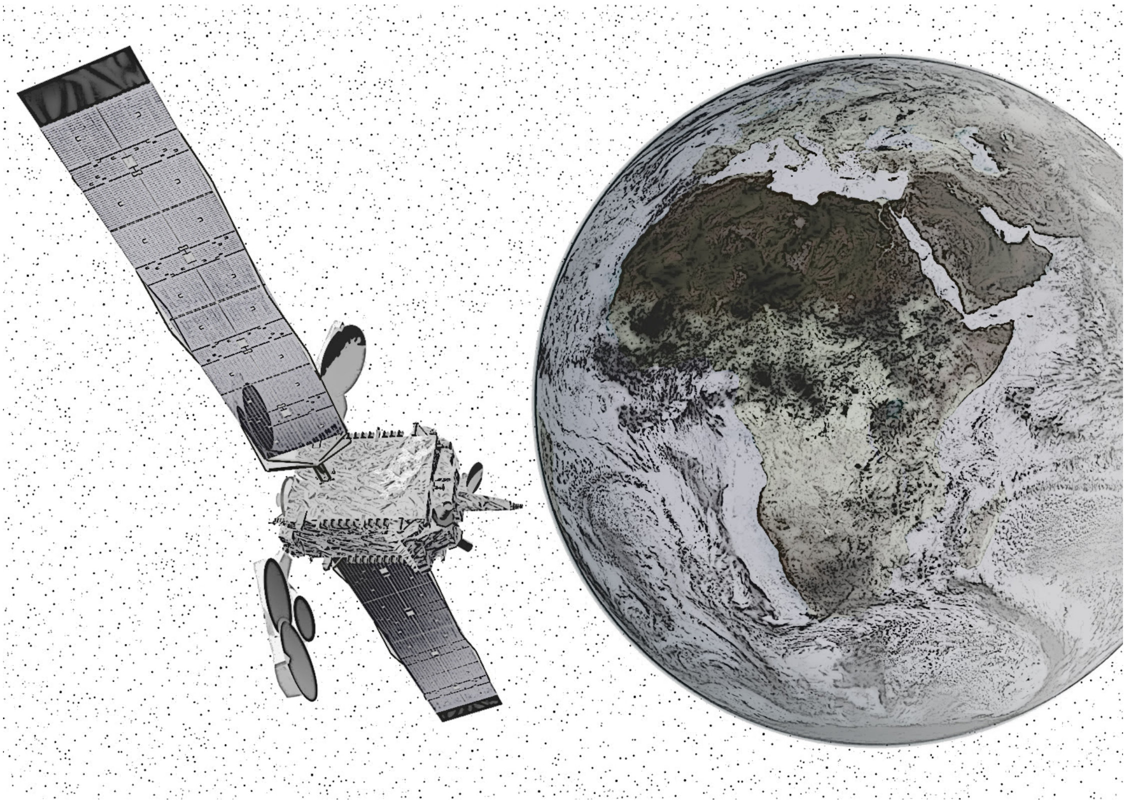

Table des matières

Sujet 1
Méthode de tir simple

1.1 Introduction

Nous allons dans cette partie voir une méthode de résolution numérique d'un problème simple de contrôle optimal utilisant les conditions nécessaires d'optimalité, c'est-à-dire le Principe du Maximum de Pontryagin (PMP). Cette méthode fait partie des méthodes dites indirectes et s'appelle la méthode de tir simple. Vous pouvez consulter la référence suivante pour une description des différentes méthodes numériques de résolution de problèmes de contrôle optimal : A. V. Rao, *A survey of numerical methods for optimal control.*

1.2 Présentation de la méthode de tir simple sur un exemple

1.2.1 Le problème de contrôle optimal

Considérons le problème de contrôle optimal suivant.

$$\text{(P1)} \qquad \left\{ \begin{array}{l} J(u(\cdot)) := \dfrac{1}{2} \displaystyle\int_0^{t_f} u(t)^2 \, \mathrm{d}t \longrightarrow \min \\[2mm] \dot{x}(t) = -x(t) + u(t), \quad u(t) \in \mathbb{R}, \quad t \in [0\,, t_f] \text{ p.p.}, \\[2mm] x(0) = x_0, \quad x(t_f) = x_f, \end{array} \right.$$

avec $t_f := 1$, $x_0 := -1$, $x_f := 0$ et $\forall\, t \in [0\,, t_f]$, $x(t) \in \mathbb{R}$. Notons

$$H(x, p, p^0, u) := p\,(-x + u) + p^0\, \frac{1}{2} u^2,$$

le pseudo-hamiltonien associé au problème (P1).

1.2.2 Application du principe du maximum de Pontryagin

D'après le PMP, si $u(\cdot)$ est une solution optimal de (P1) (avec $x(\cdot)$ la trajectoire associée) alors il existe un vecteur adjoint $p(\cdot) \in AC([0\,, t_f], \mathbb{R})$, un scalaire $p^0 \leq 0$, tels que $(p(\cdot), p^0) \neq (0, 0)$, et tels que les équations suivantes sont vérifiées pour $t \in [0\,, t_f]$ p.p. :

$$\left\{ \begin{array}{lcl} \dot{x}(t) & = & \partial_p H[t] = -x(t) + u(t), \\ \dot{p}(t) & = & -\partial_x H[t] = p(t), \\ 0 & = & \partial_u H[t] = p(t) + p^0 u(t), \end{array} \right.$$

où $[t] := (x(t), p(t), p^0, u(t))$. Il n'y a pas d'anormale car si $p^0 = 0$ alors $p(\cdot) \equiv 0$ ce qui est impossible. Ainsi, on a $p^0 < 0$.

Question 1. Expliquer pourquoi la condition de maximisation du hamiltonien donnée par le PMP :

$$H[t] = \max_{w \in \mathbb{R}} H(x(t), p(t), p^0, w),$$

est équivalente dans cet exemple à la condition :

$$0 = \partial_u H[t].$$

Notons $u_s(x, p, p^0) := -p/p^0$, la solution de l'équation $0 = \partial_u H(x, p, p^0, u)$ pour (x, p, p^0) fixé. On peut alors fixer arbitrairement $p^0 \neq 0$ car pour tout $\alpha \in \mathbb{R}^*$, $u_s(x, \alpha p, \alpha p^0) = u_s(x, p, p^0)$ et la trajectoire associée reste inchangée. Fixons $p^0 = -1$ et notons maintenant

$$u_s(x, p) = p \tag{1.1}$$

le contrôle optimal.

1.2.3 Problème aux deux bouts

L'application du PMP nous mène à résoudre le *problème aux deux bouts* (Two Points Boundary Value Problem) suivant :

$$\text{(P2)} \qquad \begin{cases} \dot{x}(t) & = -x(t) + u_s(x(t), p(t)) = -x(t) + p(t), \\ \dot{p}(t) & = \quad p(t), \\ x(0) & = x_0, \quad x(t_f) = x_f. \end{cases}$$

L'inconnue de ce problème aux deux bouts est le vecteur adjoint initial $p(0)$. En effet si l'on fixe $p(0) =: p_0$ alors d'après le théorème de Cauchy–Lipschitz, il existe une unique solution maximale $z(\cdot) := (x(\cdot), p(\cdot))$ vérifiant la dynamique (sur x et p) et la condition initiale $z(0) = (x_0, p_0)$. Le problème est donc de trouver p_0 tel que $x(t_f) = x_f$.

1.2.4 Fonction de tir et méthode de tir (simple)

On va transformer le problème aux deux bouts (P2) en un sytème d'équations non linéaires, que l'on appelle équations de tir. Pour cela, on définit tout d'abord le système hamiltonien

$$\overrightarrow{H}(x, p) := \left(\frac{\partial H}{\partial p}(x, p, u_s(x, p)), -\frac{\partial H}{\partial x}(x, p, u_s(x, p)) \right). \tag{1.2}$$

On note alors $z := (x, p)$, puis $z(\cdot, x_0, p_0)$ la solution de l'équation différentielle $\dot{z}(t) = \overrightarrow{H}(z(t))$ vérifiant $z(0, x_0, p_0) = (x_0, p_0)$. On définit enfin la *fonction de tir* suivante :

$$\begin{array}{rccl} S\colon & \mathbb{R} & \longrightarrow & \mathbb{R} \\ & y & \longmapsto & S(y) := \Pi_x(z(t_f, x_0, y)) - x_f, \end{array} \tag{1.3}$$

où Π_x est simplement la projection canonique $\Pi_x(x, p) = x$. Résoudre le problème aux deux bouts (P2) revient à trouver un zéro de la fonction de tir, *i.e.* consiste à résoudre

$$S(y) = 0. \tag{1.4}$$

La *méthode de tir simple* consiste à résoudre les équations non linéaires (1.4). On utilise pour cela une méthode de type Newton. Pour augmenter l'efficacité de l'algorithme, nous pouvons fournir la jacobienne de la fonction de tir, *cf.* sujet 2.

> *Remarque* 1.1. ✍ Si $\bar{p}_0$ vérifie $S(\bar{p}_0) = 0$, alors la courbe intégrale $\bar{z}(\cdot)$ solution de l'équation différentielle $\dot{z}(t) = \overrightarrow{H}(z(t))$, $z(0) = (x_0, \bar{p}_0)$, avec le contrôle $\bar{u}(\cdot) := u_s(\bar{z}(\cdot))$, est une BC-extrémale du problème (P1), *i.e.* cette extrémale satisfait les conditions nécessaires d'optimalité données par le PMP.

1.2.5 Calcul du zéro de la fonction de tir

Calculons la solution à la main sur cet exemple simple.

$$\dot{p}(t) = p(t) \Longrightarrow p(t) = e^t\, p_0,\ p_0 := p(0) \Longrightarrow x(t) = (0.5\, p_0\, e^{2\,t} + C)\, e^{-t},\ C \in \mathbb{R}.$$

Or $x(0) = x_0$ donc

$$x(t) = (0.5\, p_0\, (e^{2\,t} - 1) + x_0)\, e^{-t}$$

et finalement, puisque $x_0 = -1$, $x(t_f) = x_f = 0$ et $t_f = 1$, on a

$$\bar{p}_0 = \frac{2\,(x_f\, e^{t_f} - x_0)}{e^{2\,t_f} - 1} = \frac{2}{e^2 - 1} \approx 0.313.$$

1.3 Implémentation Matlab de la méthode de tir simple

L'objectif est d'étudier le problème (P1) pour retrouver les résultats de la figure 1.1.

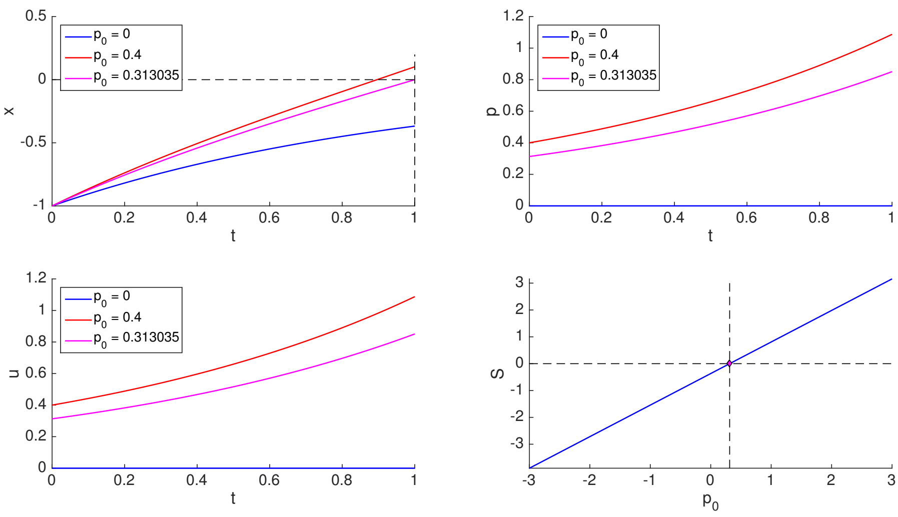

FIGURE 1.1 – Résultats après exécution du fichier `sujet1_tir_simple/probleme1/main.m`.

1.3.1 Le système hamiltonien

▷ **Exercice 1.1**. Implémentation du système hamiltonien.

1. Se rendre dans le répertoire `sujet1_tir_simple/probleme1`.
2. Compléter le fichier `lib/control.m` qui code le contrôle, *cf.* éq. (1.1).
3. Compléter le fichier `lib/hvfun.m` qui code le système hamiltonien, *cf.* éq. (1.2).
4. Lancer le script `main.m` et vérifier le code de `lib/hvfun.m`.

1.3.2 Intégration numérique du système hamiltonien

▷ **Exercice 1.2**. Intégration numérique du système hamiltonien.

1. Compléter le fichier `lib/exphvfun.m` qui permet le calcul de la solution de l'équation différentielle $\dot{z}(t) = \overrightarrow{H}(z(t))$, $z(0) = z_0$.
   - Utiliser un "Function Handle", voir "Creating a Function Handle" sur la documentation de Matlab.
   - Utiliser la fonction Matlab `ode45.m` (voir » `doc ode45`).

   Attention, les fonctions `ode45` et `exphvfun` ne renvoient pas les mêmes types de données en sortie.
2. Lancer le script `main.m` et vérifier le code de `lib/exphvfun.m`.

Remarque 1.2. La notation `exphvfun` vient de la notation mathématique suivante :

$$\exp(t_f \overrightarrow{H})(z_0) \coloneqq z(t_f, z_0),$$

où $z(\cdot, z_0)$ est la solution de l'équation différentielle $\dot{z}(t) = \overrightarrow{H}(z(t))$, $z(0) = z_0$.

1.3.3 Fonction de tir

▷ **Exercice 1.3**. Implémentation de la fonction de tir.

1. Compléter le fichier `lib/sfun.m` qui code la fonction de tir (1.3).
2. Lancer le script `main.m` et vérifier le code de `lib/sfun.m`.

1.3.4 Méthode de tir simple

▷ **Exercice 1.4**. Implémentation de la méthode de tir et résolution des équations de tir.

1. Compléter le fichier `lib/ssolve.m` qui code la méthode de tir, pour résoudre l'équation (1.4). Utiliser la fonction Matlab `fsolve.m` (voir » `doc fsolve`) et un "Function Handle".
2. Lancer le script `main.m` et vérifier le code de `lib/ssolve.m`.

1.4 Convergence de la méthode de tir

L'objectif est d'analyser une cause de non convergence de la méthode de tir et de retrouver les résultats de la figure 1.2.

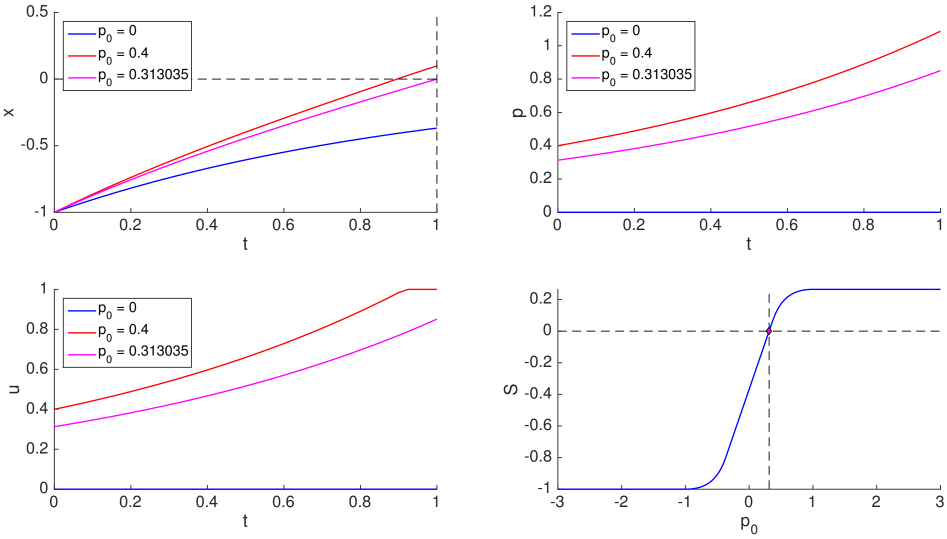

FIGURE 1.2 – Résultats après exécution du fichier `sujet1_tir_simple/probleme2/main.m`.

1.4.1 Problème avec contraintes sur le contrôle

On considère le problème de contrôle (P1) où l'on remplace la contrainte sur le contrôle par $u(t) \in [-1\,,1]$. La condition de maximisation nous donne comme loi de contrôle :

$$u(t) = \begin{cases} u_s(x(t), p(t)) = p(t) & \text{si } |p(t)| \leq 1, \\ +1 & \text{si } p(t) > 1, \\ -1 & \text{si } p(t) < -1. \end{cases} \tag{1.1*}$$

1.4.2 Résolution des équations de tir et influence du point initial

▷ **Exercice 1.5**. Résolution des équations de tir.

1. Se rendre dans le répertoire `sujet1_tir_simple/probleme2`.
2. Implémenter dans le répertoire `lib` les fonctions `control` (qui code (1.1*)), `hvfun`, `exphvfun`, `sfun` et `ssolve`.
3. Exécuter le script `main.m` et vérifier les résultats.
4. Vérifier que la méthode de tir ne converge pas si l'on donne comme point initial à l'algorithme : $y^{(0)} = 1.5$.

Question 2. Expliquer pourquoi la méthode de tir ne converge pas pour $y^{(0)} = 1.5$.

1.5 Problème en dimension 2 avec contraintes sur le contrôle

L'objectif est dans un premier temps, de retrouver les résultats de la figure 1.3, puis d'étudier l'influence des contraintes sur le contrôle pour un problème avec un critère quadratique en le contrôle.

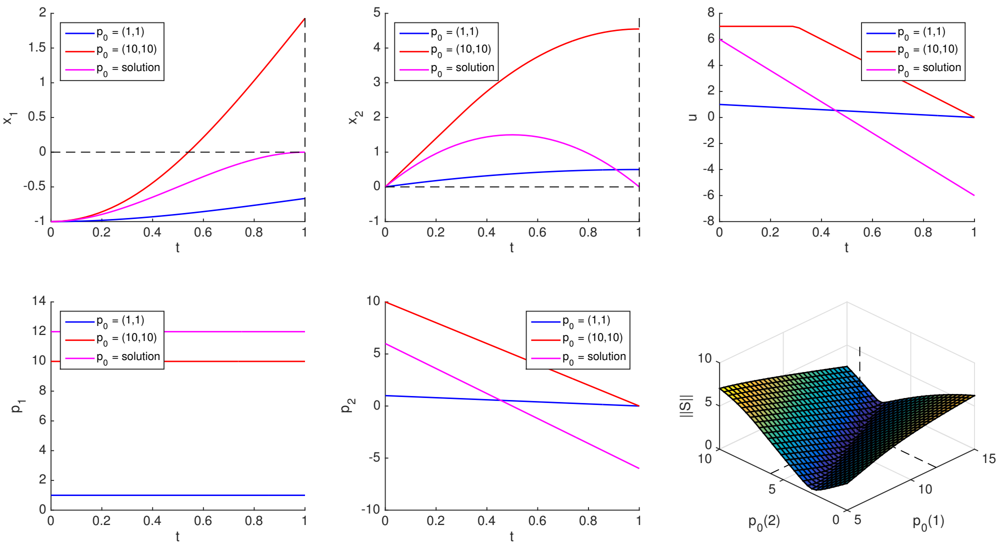

FIGURE 1.3 – Résultats après exécution du fichier `sujet1_tir_simple/probleme3/main.m`.

1.5.1 Définition du problème

Considérons le problème de contrôle optimal suivant.

$$
\text{(P3)} \qquad \left\{
\begin{array}{l}
J(u(\cdot)) \coloneqq \dfrac{1}{2}\displaystyle\int_0^{t_f} u(t)^2 \,\mathrm{d}t \longrightarrow \min \\[2mm]
\dot{x}_1(t) = x_2(t) \\[2mm]
\dot{x}_2(t) = u(t), \quad |u(t)| \le u_{\max}, \quad t \in [0\,, t_f] \text{ p.p.}, \\[4mm]
x(0) = x_0, \quad x(t_f) = x_f,
\end{array}
\right.
$$

avec $t_f \coloneqq 1$, $x_0 \coloneqq (-1, 0)$, $x_f \coloneqq (0, 0)$ et $\forall\, t \in [0\,, t_f]$, $x(t) = (x_1(t), x_2(t)) \in \mathbb{R}^2$. Notons

$$
H(x, p, p^0, u) \coloneqq p_1\, x_2 + p_2\, u + p^0\, \frac{1}{2} u^2, \quad x = (x_1, x_2), \quad p = (p_1, p_2),
$$

le pseudo-hamiltonien associé au problème (P3). On considère le cas normal et on fixe $p^0 = -1$. La condition de maximisation nous donne comme loi de contrôle :

$$
u(t) = \left\{
\begin{array}{ll}
u_s(x(t), p(t)) \coloneqq p_2(t) & \text{si } |p_2(t)| \le u_{\max}, \\[2mm]
+u_{\max} & \text{si } p_2(t) > u_{\max}, \\[2mm]
-u_{\max} & \text{si } p_2(t) < -u_{\max}.
\end{array}
\right. \tag{1.5}
$$

1.5.2 Résolution des équations de tir et influence de la borne sur le contrôle

▷ **Exercice 1.6**. Résolution des équations de tir.

1. Se rendre dans le répertoire `sujet1_tir_simple/probleme3`.
2. Implémenter dans le répertoire `lib` les fonctions `control`, `hvfun`, `exphvfun`, `sfun` et `ssolve`, associées au problème (P3).
3. Exécuter le script `main.m` et vérifier les résultats. La solution des équations de tir $S(y) = 0$ (y est ici le vecteur adjoint initial p_0) associées au problème (P3) est $\bar{p}_0 = (12, 6)$ pour $u_{\max} \geq 6$.
4. Exécuter le script `main.m` avec $u_{\max} = 5$ et vérifier que $\bar{p}_0 = (12.8977, 6.44364)$, et qu'il existe $0 \leq t_1 \leq t_2 \leq t_f$ tels que la loi de commande solution $t \mapsto \bar{u}(t)$ vérifie : $\bar{u}(t) = u_{\max}$ sur $[0\,, t_1]$, $\bar{u}(t) = u_s(\bar{x}(t), \bar{p}(t))$ sur $[t_1\,, t_2]$ et $\bar{u}(t) = -u_{\max}$ sur $[t_2\,, t_f]$.

Remarque 1.3. ✍ La loi de commande $\bar{u}(\cdot)$ associée à la solution des équations de tir est lisse lorsque $u_{\max} \geq 6$. Pour $u_{\max} < 6$, la loi de commande perd en régularité. Pour $u_{\max} = 5$, elle est continue mais pas C^1, seulement C^1 par morceaux (elle est même lisse par morceaux).

Remarque 1.4. ✍ On verra au sujet 4 des lois de commande qui ne sont pas continues, mais toujours lisses par morceaux. Attention, dans ce cas là, nous n'utiliserons plus une méthode de tir simple mais de tir multiple pour résoudre les équations de tir.

Sujet 2
Jacobienne de la fonction de tir simple

2.1 Sur l'intérêt du calcul de la jacobienne de la fonction de tir

On considère le problème de contrôle optimal suivant :

$$(\mathrm{P4}_\varepsilon) \qquad \left\{ \begin{array}{l} J_\varepsilon(u(\cdot)) := \displaystyle\int_0^{t_f} \Big(|u(t)| - \varepsilon \left(\ln(|u(t)|) + \ln(1 - |u(t)|) \right)\Big) \, \mathrm{d}t \longrightarrow \min \\ \dot{x}(t) = -x(t) + u(t), \quad |u(t)| \leq 1, \quad t \in [0\,, t_f] \text{ p.p.}, \\ x(0) = x_0, \quad x(t_f) = x_f, \end{array} \right.$$

avec $\varepsilon \in [0\,, 1]$, $t_f := 2$, $x_0 := 0$, $x_f := 0.5$ et $\forall\, t \in [0\,, t_f]$, $x(t) \in \mathbb{R}$. Notons

$$H(x, p, p^0, u) := p\,(-x + u) + p^0 \Big(|u(t)| - \varepsilon \left(\ln(|u(t)|) + \ln(1 - |u(t)|) \right)\Big),$$

le pseudo-hamiltonien associé au problème ($\mathrm{P4}_\varepsilon$). On considère le cas normal et on fixe $p^0 = -1$. La condition de maximisation du hamiltonien donne ici

$$u_\varepsilon(p) := \left\{ \begin{array}{ll} \dfrac{-2\,\varepsilon \,\mathrm{sign}(p)}{\psi(p) - 2\,\varepsilon - \sqrt{\psi(p)^2 + 4\,\varepsilon^2}} & \text{si } p \neq 0, \\ \dfrac{\pm 2\,\varepsilon}{-1 - 2\,\varepsilon - \sqrt{1 + 4\,\varepsilon^2}} & \text{si } p = 0, \end{array} \right. \tag{2.1}$$

où $\psi(p) = -1 + |p|$.

Remarque 2.1. Dans le code on prendra $u_\varepsilon(0) = -2\,\varepsilon\,/(-1 - 2\,\varepsilon - \sqrt{1 + 4\,\varepsilon^2})$.

▷ **Exercice 2.1**. Sensibilité de la méthode de tir et jacobienne de la fonction de tir.

1. Se rendre dans le répertoire `sujet2_jacobienne/partie1`.
2. Implémenter dans le répertoire `lib` les fonctions `control`, `hvfun`, `exphvfun`, `sfun` et `ssolve`, associées au problème ($\mathrm{P4}_\varepsilon$).
3. Exécuter le script `main.m` et vérifier les résultats, *i.e.* comparer à la figure 2.1.
4. Vérifier que pour $\varepsilon = 0.01$, la méthode de tir $S(y) = 0$ associées au problème ($\mathrm{P4}_\varepsilon$) converge si l'on donne comme point de départ $y^{(0)} = 0.37$ ou $y^{(0)} = 0.39$, mais ne converge pas pour $y^{(0)} = 0.38$.
5. Vérifier que la méthode de tir (pour $\varepsilon = 0.01$) converge si l'on donne comme point de départ $y^{(0)} = 0.38$, si l'on prend comme erreur locale d'intégration les valeurs `RelTol` $=$ `Abstol` $= 1e^{-6}$ (voir » `doc ode45` et » `doc odeset`).

✍ Nous allons voir que le problème de convergence vient du calcul de la jacobienne de la fonction de tir (qui ici est calculée par différences finies).

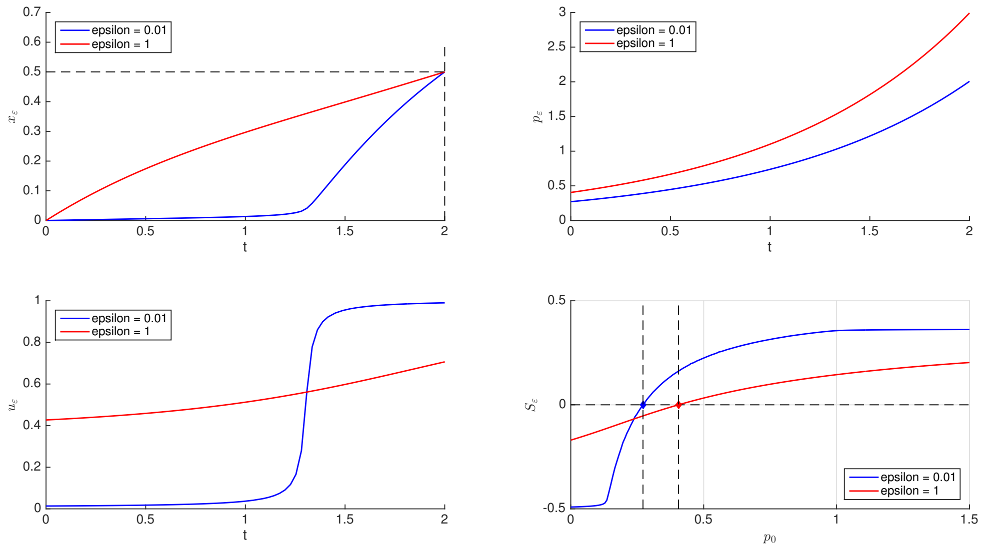

FIGURE 2.1 – Résultats après exécution du fichier `sujet2_jacobienne/partie1/main.m`.

2.2 Introduction aux différences finies

2.2.1 Rappels

Considérons deux espaces de Banach $(E, \|\cdot\|_E)$ et $(F, \|\cdot\|_F)$. Soit $g\colon E \to F$ une application. On rappelle, pour $h \in E$, la notation de Landau $o(h)$:

$$g(h) = o(\|h\|_E) \quad \Longleftrightarrow \quad \exists\, \varepsilon\colon E \to F, \text{ tel que } g(h) = \|h\|_E\, \varepsilon(h) \text{ et } \lim_{\|h\|_E \to 0} \|\varepsilon(h)\|_F = 0.$$

Définition 2.1 (Différentielle de Fréchet). Soient une application $f\colon \Omega \subset E \to F$, Ω ouvert et un point $x \in \Omega$. On dit que f est *différentiable* au point x ssi il existe une application $T_{f,x} \in \mathcal{L}(E, F)$ telle que pour tout $h \in E$ vérifiant $x + h \in \Omega$, on ait

$$f(x+h) - f(x) - T_{f,x}(h) = o(\|h\|_E).$$

L'application linéaire continue $T_{f,x}$, si elle existe est unique, et on l'appelle la *différentielle de Fréchet* de f au point x.

> *Remarque* 2.2. On utilisera la notation $\mathrm{d}f(x) \cdot h \coloneqq T_{f,x}(h)$ ou $f'(x) \cdot h \coloneqq T_{f,x}(h)$.

Si f est différentiable en x et si h est un vecteur de E, on a

$$\mathrm{d}f(x) \cdot h = \lim_{t \to 0} \frac{f(x + t\, h) - f(x)}{t},$$

t étant une variable réelle. Ainsi, $\mathrm{d}f(x) \cdot h$ est égal au vecteur vitesse $\frac{\mathrm{d}}{\mathrm{d}t} f(x + t\, h)|_{t=0}$, appelée *dérivée directionnelle* de f en x dans la direction h.

2.2.2 Différences finies

Soient f une fonction lisse de $\mathbb{R}^n$ dans $\mathbb{R}^m$, x un point de $\mathbb{R}^n$ et h un vecteur de $\mathbb{R}^n$. On note $g\colon t \mapsto f(x + t\,h)$. Pour t proche de 0, on a d'après la formule de Taylor-Young :

$$g(t) = \sum_{k=0}^{n} \frac{t^k}{k!}\, g^{(k)}(0) + R_n(t), \quad R_n(t) = o(t^n),$$

et d'après l'inégalité de Taylor-Lagrange,

$$|R_n(t)| \leq \frac{M_n\,|t|^{n+1}}{(n+1)!},$$

où M_n est une constante positive majorant la dérivée à l'ordre $n+1$. De même,

$$g(-t) = \sum_{k=0}^{n} \frac{(-t)^k}{k!}\, g^{(k)}(0) + o(t^n).$$

Formule des différences finies avants. La méthode des différences finies avants consiste à approcher la différentielle de f en x dans la direction de h par la formule suivante :

$$\frac{f(x+t\,h) - f(x)}{t} = \frac{g(t) - g(0)}{t} = g'(0) + \frac{t}{2}\, g''(0) + \frac{t^2}{3!}\, g^{(3)}(0) + o(t^2). \tag{2.2}$$

On obtient ainsi une approximation de $g'(0) = \mathrm{d}f(x)\cdot h$ **d'ordre 1** si $g''(0) \neq 0$ ou au moins d'ordre 2 sinon. Notons $\mathrm{num}(g,t)$ la valeur $g(t)$ calculée numériquement et supposons que l'on puisse majorée l'erreur relative du calcul numérique par :

$$|\mathrm{num}(g,t) - g(t)| \leq \theta L_f,$$

où L_f est une borne de la valeur de $g(\cdot)$ sur le domaine d'intérêt. Ainsi, pour $t > 0$ petit, on a :

$$\begin{aligned}
\left| \frac{\mathrm{num}(g,t) - \mathrm{num}(g,0)}{t} - g'(0) \right| &= \left| \frac{g(t) + e_1 - g(0) - e_2}{t} - g'(0) \right| \\
&= \left| \frac{R_1(t)}{t} + \frac{e_1 - e_2}{t} \right| \\
&\leq \frac{M_1\,t}{2} + 2\frac{\theta L_f}{t} =: \phi(t).
\end{aligned}$$

La fonction ϕ atteint son minimum en la valeur

$$\bar{t} = 2\sqrt{\frac{\theta L_f}{M_1}},$$

et si on suppose que le ratio L_f/M_1 est d'un ordre de grandeur peu élevé, alors le choix suivant pour t est presque optimal :

$$\bar{t} = \sqrt{\theta},$$

et on obtient ainsi une approximation de l'erreur totale $\phi(\bar{t})$ proportionnelle à $\sqrt{\theta}$.

Formule des différences finies centrées. La méthode des différences finies centrées consiste à approcher la différentielle de f en x dans la direction de h par la formule suivante :

$$\frac{f(x+t\,h)-f(x-t\,h)}{2\,t} = \frac{g(t)-g(-t)}{t} = g'(0) + \frac{t^2}{3!}\,g^{(3)}(0) + \frac{t^4}{5!}\,g^{(5)}(0) + o(t^4). \qquad (2.3)$$

On obtient ainsi une approximation de $g'(0) = \mathrm{d}f(x)\cdot h$ **d'ordre 2** si $g^{(3)}(0) \neq 0$ ou au moins d'ordre 4 sinon.

> *Remarque* 2.3. ✍ Les différences finies centrées sont d'ordre plus élevé mais demandent plus d'évaluations de la fonction f.

> **Question 3**. Sachant que l'on commet une erreur numérique lors du calcul de g en un point t, quel est le choix optimal pour t pour approcher la valeur de $g'(0)$ par différences finies centrées ? Quelle est alors la valeur de la borne de l'erreur totale ?

▷ **Exercice 2.2**. Approximation par différences finies avants.

1. Se rendre dans le répertoire `sujet2_jacobienne/partie2`.
2. Implémenter la fonction `finiteDiff` dans le fichier éponyme du répertoire `lib`, qui approche par différences finies avants, la dérivée directionnelle de f en x dans la direction h.
3. Exécuter le script `mainTestFD.m` et vérifier que la valeur optimale de t est de l'ordre de $\sqrt{\varepsilon_{\mathrm{mach}}}$ où $\varepsilon_{\mathrm{mach}}$ est l'epsilon machine. Ici, $f(x) = -\cos(x)$.

2.3 Jacobienne de la fonction de tir

2.3.1 Calcul de la jacobienne de la fonction de tir

Voyons le calcul de la jacobienne de la fonction de tir sur un exemple. Donnons la jacobienne de la fonction de tir associée au problème ($P4_\varepsilon$). Pour ce problème, la fonction de tir s'écrit

$$S(y) = \Pi_x(z(t_f, x_0, y)) - x_f = x(t_f, x_0, y) - x_f,$$

avec Π_x la projection canonique $\Pi_x(x,p) = x$ et où $z(\cdot, z_0)$ est solution de $\dot{z}(t) = \overrightarrow{H}(z(t))$, $z(0, z_0) = z_0$, avec la notation habituelle $z_0 = (x_0, p_0)$. Ainsi, on a :

$$\mathrm{d}S(y)\cdot h = \frac{\partial x}{\partial p_0}(t_f, x_0, y)\cdot h = \frac{\partial x}{\partial z_0}(t_f, x_0, y)\cdot(0,h) = \Pi_x\left(\frac{\partial z}{\partial z_0}(t_f, x_0, y)\cdot(0,h)\right). \qquad (2.4)$$

On voit alors que le calcul de la jacobienne de S fait intervenir

$$\frac{\partial z}{\partial p_0}(t_f, x_0, y) = \frac{\partial z}{\partial z_0}(t_f, x_0, y)\cdot(0_n, I_n),$$

où 0_n est la matrice nulle de taille n, I_n est la matrice identité, et n est la dimension de l'état. Enfin, d'après le cours sur les équations différentielles ordinaires [1], $\frac{\partial z}{\partial p_0}(\cdot, x_0, y)$ est solution des *équations variationnelles* suivantes :

$$\dot{\widehat{\delta z}}(t) = \mathrm{d}\overrightarrow{H}(z(t, x_0, y))\cdot\delta z(t), \quad \delta z(0) = \begin{bmatrix} 0_n \\ I_n \end{bmatrix}. \qquad (2.5)$$

1. J. Gergaud, Équations différentielles ordinaires.

Remarque 2.4. Si l'on écrit

$$z(t, x_0, p_0) = (x_0, p_0) + \int_0^t \overrightarrow{H}(z(s, x_0, p_0))\,\mathrm{d}s,$$

et si $z(\cdot, x_0, p_0)$ est dérivable par rapport à la condition initiale, alors on retrouve le résultat en dérivant :

$$\frac{\partial z}{\partial p_0}(t, x_0, p_0) = (0_n, I_n) + \int_0^t \mathrm{d}\overrightarrow{H}(z(s, x_0, p_0)) \cdot \frac{\partial z}{\partial p_0}(s, x_0, p_0)\,\mathrm{d}s.$$

L'objectif dans un premier temps est de retrouver les résultats de la figure 2.2.

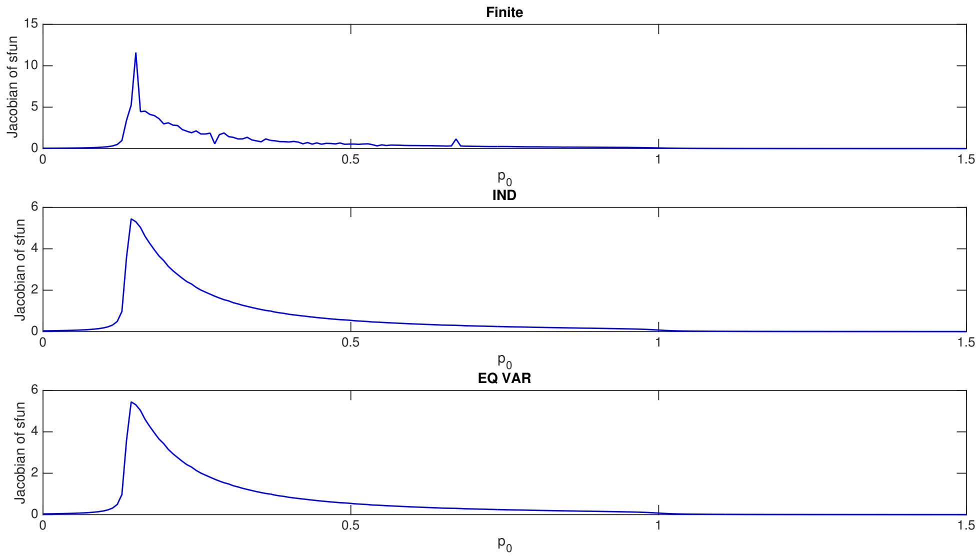

FIGURE 2.2 – Résultats après exécution du fichier `sujet2_jacobienne/partie2/mainDiff.m`.

2.3.2 Différences finies (externes)

Remarque 2.5. Attention, la fonction `ssolve` est déjà écrite.

▷ **Exercice 2.3**. Approximation de la jacobienne de la fonction de tir par différences finies.

1. Implémenter dans le répertoire `lib` de `sujet2_jacobienne/partie2`, les fonctions `control`, `hvfun`, `exphvfun`, `sfun` associées au problème ($P4_\varepsilon$).
2. Utiliser la fonction `finiteDiff` pour compléter la partie calcul par différences finies (`finite`) dans le fichier `lib/sjac.m`.
3. Exécuter le script `mainDiff.m` et vérifier les résultats, *cf.* figure 2.2.

2.3.3 Différentiation interne de Bock

Nous allons approcher par différence finie le deuxième membre de l'équation variationnelle (2.5) [2]. On a alors, en notant $z(t) := z(t, x_0, y)$,

$$\mathrm{d}\overrightarrow{H}(z(t)) \cdot \delta z(t) \approx \frac{\overrightarrow{H}(z(t) + \eta\, \delta z(t)) - \overrightarrow{H}(z(t))}{\eta},$$

avec $\eta > 0$ petit. On peut prendre η de l'ordre de la racine carrée de l'epsilon machine.

▷ **Exercice 2.4**. Approximation de la jacobienne de la fonction de tir par différences finies interne (de Bock).

1. Compléter les parties calcul par différences finies internes (`ind`) dans les fichiers `lib/dhvfun.m` et `lib/sjac.m`. Utiliser la fonction `expdhvfun` qui est déjà codée dans le fichier `lib/expdhvfun.m`.
2. Exécuter le script `mainDiff.m` et vérifier les résultats, *cf.* figure 2.2.

2.3.4 Équations variationnelles

Nous allons directement résoudre les équations variationnelles (2.5) pour calculer la jacobienne de la fonction de tir. Pour cela, nous donnons pour $p \neq 0$,

$$u'_\varepsilon(p) = \frac{2\,\varepsilon \left(1 - \dfrac{\psi(p)}{\sqrt{\psi(p)^2 + 4\,\varepsilon^2}}\right)}{\left(\psi(p) - 2\,\varepsilon - \sqrt{\psi(p)^2 + 4\,\varepsilon^2}\right)^2}.$$

Bien que la fonction ne soit pas définie pour $p = 0$, et donc non dérivable, on prendra dans le code la même expression.

▷ **Exercice 2.5**. Calcul de la jacobienne de la fonction de tir via les équations variationnelles.

1. Compléter le fichier `lib/dcontrol.m` qui code la fonction u'_ε.
2. Compléter les parties calcul via les équations variationnelles (`eqvar`) dans les fichiers `lib/dhvfun.m` et `lib/sjac.m`.
3. Exécuter le script `mainDiff.m` et vérifier les résultats, *cf.* figure 2.2.

2. H.G. Bock. *Numerical treatment of inverse problems in chemical reaction kinetics*, in K. H. Hebert, P. Deuflhard, and W. Jager, editors, Modelling of chemical reaction systems, volume 18 of Springer series in Chem. Phys., pages 102–125, 1981.

2.4 Résolution des équations de tir

▷ **Exercice 2.6**. Résolution des équations de tir : exécution du script `main.m`.

1. Vérifier que la méthode de tir ne converge pas avec le calcul de la jacobienne par différences finies, avec les tolérances d'intégration numérique par défaut et un pas de différences finies de l'ordre de l'epsilon machine (modifier `lib/sjac.m` si nécessaire). On retrouve alors le comportement par défaut vu en section 2.1.
2. Vérifier que la méthode converge si l'on prend les tolérances par défaut mais que l'on choisit un pas adapté, de l'ordre de la racine carrée de l'erreur d'intégration numérique (modifier `lib/sjac.m` si nécessaire). On rappelle que les tolérances par défaut données par `odeset` sont `AbsTol` $= 1e^{-6}$ et `Reltol` $= 1e^{-3}$.
3. Vérifer que la méthode (`ssolve`) converge si l'on calcule la jacobienne avec l'option `ind` ou `eqvar`.

Remarque 2.6. ✍ On peut voir sur la figure 2.3, le comportement de la fonction `fsolve` de Matlab vis à vis de l'utilisation des différences finies pour le calcul approché de la jacobienne. Par défaut, le pas est de l'ordre de la racine carrée de l'epsilon machine pour les différences finies avants et de la racine cubique pour les différences centrées.

`FiniteDifferenceStepSize`	Scalar or vector step size factor for finite differences. When you set `FiniteDifferenceStepSize` to a vector v, forward finite differences steps `delta` are
	`delta = v.*sign′(x).*max(abs(x),TypicalX);`
	where `sign′(x) = sign(x)` except `sign′(0) = 1`. Central finite differences are
	`delta = v.*max(abs(x),TypicalX);`
	Scalar `FiniteDifferenceStepSize` expands to a vector. The default is `sqrt(eps)` for forward finite differences, and `eps^(1/3)` for central finite differences.

`FiniteDifferenceType`	Finite differences, used to estimate gradients, are either `'forward'` (default), or `'central'` (centered). `'central'` takes twice as many function evaluations, but should be more accurate.

Figure 2.3 – Extrait de la documentation de Matlab sur les différences finies avec `fsolve`.

Sujet 3
Méthode de continuation discrète

On considère le problème de contrôle optimal (P4$_\varepsilon$). L'objectif est de retrouver les résultats des figures 3.1 et 3.2 et de déterminer la structure de la solution lorsque $\varepsilon \to 0$.

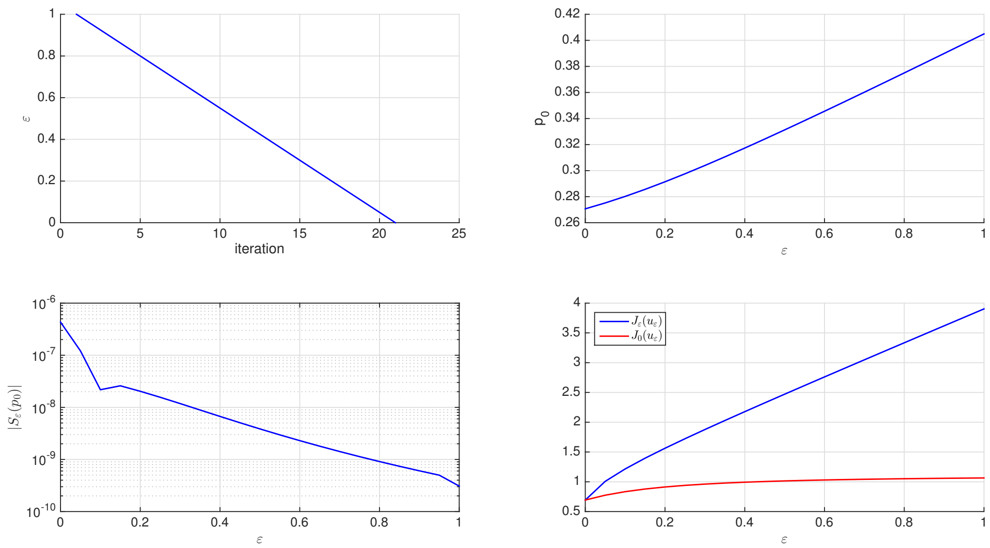

Figure 3.1 – Figure 1 après exécution du fichier `sujet3_continuation/main.m`.

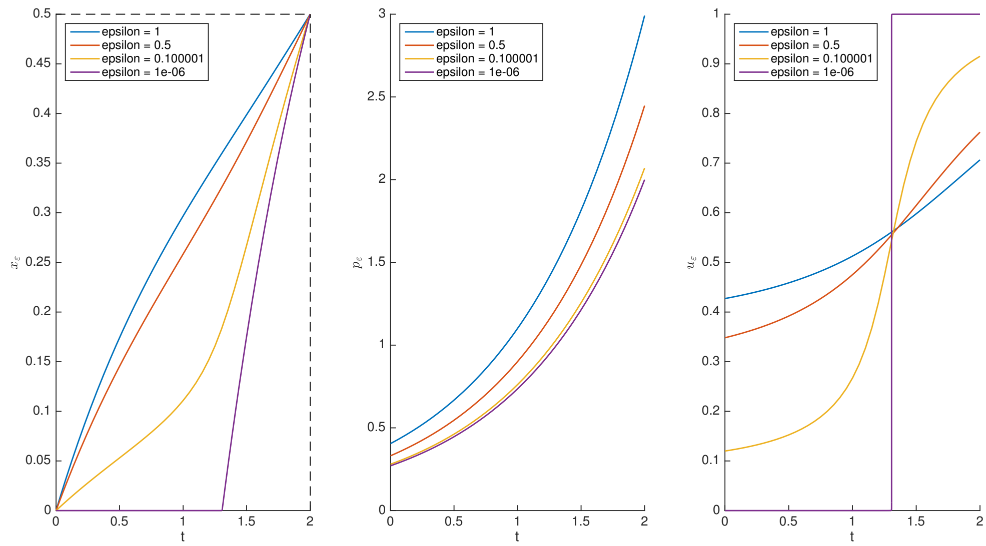

Figure 3.2 – Figure 2 après exécution du fichier `sujet3_continuation/main.m`.

▷ **Exercice 3.1**. Continuation discrète sur le paramètre ε et détermination de la structure lorsque $\varepsilon \to 0$.

1. Se rendre dans le répertoire `sujet3_continuation`.
2. Implémenter dans le répertoire `lib` les fonctions `control`, `dcontrol`, `hvfun`, `dhvfun`, `exphvfun`, `expdhvfun`, `sfun`, `sjac`, `finiteDiff` et `ssolve`, associées au problème (P4_ε) (voir `sujet2_jacobienne/partie2/lib`).
3. Compléter le fichier `expcost.m` permettant de calculer la valeur du critère.
4. Jouer sur les paramètres (dans le fichier `main.m`) de la méthode `lib/continuation` (voir `» help continuation`) et exécuter le script `main.m` pour retrouver les résultats des figures 3.1 et 3.2.

Remarque 3.1. On note γ_+ un arc tel que $u(\cdot) \equiv +1$ tout au long de l'arc, γ_- si $u(\cdot) \equiv -1$ et γ_0 si $u(\cdot) \equiv 0$. On note $\gamma_1\gamma_2$ un arc γ_1 suivi d'un arc γ_2.

Question 4. Donner en fonction de γ_+, γ_- et/ou γ_0, la structure de la BC-extrémale limite lorsque $\varepsilon \to 0$ pour le problème (P4_ε).

Sujet 4
Méthode de tir multiple et contrôle bang-bang

On considère le problème de contrôle optimal $(\mathrm{P4}_\varepsilon)_{|\varepsilon=0}$. Dans ce cas, le contrôle optimal possède une discontinuité et nous allons donc utiliser une méthode dite de tir multiple. On introduit le paramètre `iarc` dans les fonctions `control`, `hvfun`, `exphvfun` et `expcost` qui correspond à l'indice de l'arc courant. On notera $y = (p_0, t_1, z_1)$ l'inconnue de la fonction de tir, où p_0 est le vecteur adjoint initial, t_1 est l'instant de transition entre l'arc 1 et l'arc 2 et où z_1 est la valeur de l'état et état adjoint à cet instant. La fonction de tir $S\colon \mathbb{R}^4 \to \mathbb{R}^4$ aura une équation permettant d'atteindre la cible x_f, une équation permettant la commutation entre les arcs 1 et 2, et une dernière équation (dite de raccordement) assurant la continuité de l'état et de l'état adjoint à l'instant t_1.

Remarque 4.1. Attention, les commandes `expcost` pour le calcul du coût et `ssolve` sont déjà écrites !

Question 5. Donner le pseudo-hamiltonien associé au problème $(\mathrm{P4}_\varepsilon)_{|\varepsilon=0}$ et déterminer la fonction de commutation Φ permettant la transition entre arcs $\gamma_\pm$ et γ_0.

Question 6. Déterminer la fonction de tir S.

▷ **Exercice 4.1**. Tir multiple et contrôle bang-bang.

1. Se rendre dans le répertoire `sujet4_bang_bang`.
2. Implémenter dans le répertoire `lib` les fonctions `control`, `hvfun`, `exphvfun`, `sfun`, `finiteDiff` et `sjac` (seulement la partie par différences finies). Attention, on a introduit l'argument `iarc` !
3. Utiliser la BC-extrémale du problème $(\mathrm{P4}_\varepsilon)$ avec ε petit (voir sujet 3) pour initialiser la méthode de tir (dans le fichier `main.m`) et résoudre le système d'équations $S(y) = 0$.
4. Retrouver comme solution $p_0 \approx 0.270671$, $t_1 \approx 1.30685$ et $z_1 = (0, 1)$ et vérifier que la valeur du critère est bien la valeur limite donnée par la figure 3.1.

Question 7. Déterminer la jacobienne de la fonction de tir.

▷ **Exercice 4.2**. Résoudre le système d'équations $S(y) = 0$ en utilisant l'IND de Bock pour le calcul de la jacobienne de la fonction de tir.

Sujet 5
Méthodes de tir simple et multiple avec HamPath

5.1 Présentation de HamPath

5.1.1 Introduction

Le code HamPath est développé par J.-B. Caillau, Prof. à l'Institut de Mathématiques de Bourgogne (Dijon), O. Cots et J. Gergaud. Ce code permet :

- de résoudre les problèmes de contrôle optimal à commandes continues via la méthode du tir simple ;
- de résoudre les problèmes de contrôle optimal à commandes non continues via la méthode du tir multiple ;
- de résoudre les familles de problèmes de contrôle optimal à un paramètre via la méthode homotopique (hors programme) ;
- de calculer les conditions du deuxième ordre via les points conjugués (hors programme).

5.1.2 Schéma général de HamPath

L'utilisateur fournit à HamPath en Fortran la fonction de tir $S(y)$ et le hamiltonien maximisé $h(z) := \max_u H(z, p^0, u)$, et après compilation de ces routines, HamPath fournit une liste de fonctions Matlab : `hfun`, `hvfun`, `exphvfun`, `dhvfun`, `expdhvfun`, `sfun`, `sjac`, `ssolve`, `hampath`...

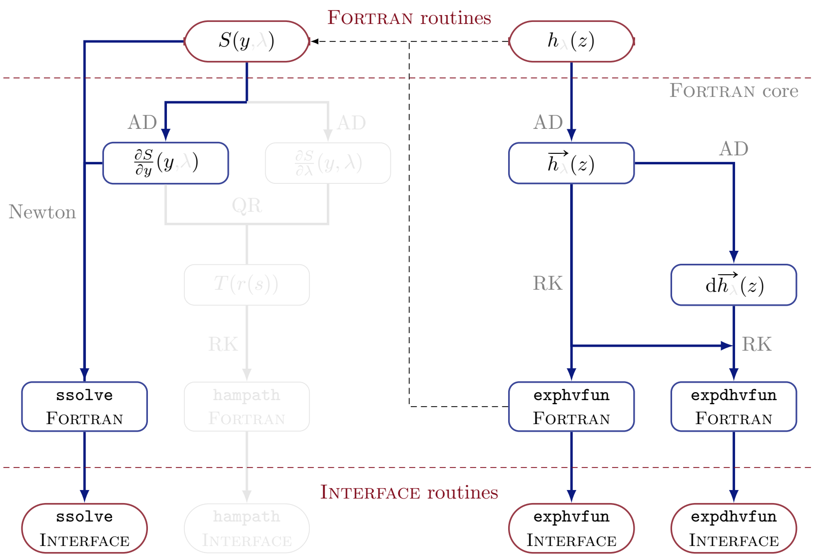

Figure 5.1 – Vue schématique de HamPath.

5.2 Tir simple et `HamPath`

5.2.1 Exemple

Considérons l'exemple de tir simple de la documentation (disponible sous Moodle) de `HamPath` :

$$(P5) \quad \begin{cases} J(u(\cdot)) = \dfrac{1}{2}\displaystyle\int_{t_0}^{t_f} u(t)^2 \, \mathrm{d}t \longrightarrow \min \\[2ex] \dot{x}(t) = v(t), \\ \dot{v}(t) = -\lambda v(t)^2 + u(t), \quad u(t) \in \mathbb{R}, \quad t \in [t_0 \, , t_f] \text{ p.p.}, \\[2ex] x(t_0) = x_0, \quad x(t_f) = x_f, \\ v(t_0) = v_0, \quad v(t_f) = v_f, \end{cases}$$

avec $t_0 = 0$, $t_f = 1$, $q_0 \coloneqq (x_0, v_0) = (-1, 0)$ et $q_f \coloneqq (x_f, v_f) = (0, 0)$. On définit le vecteur de paramètres $par \coloneqq (t_0, t_f, x_0, v_0, x_f, v_f, \lambda)$.

5.2.1.1 Implémentation par l'utilisateur de la routine Fortran `hfun`

> *Remarque* 5.1. La variable `iarc` a le même rôle que lors des TP sous Matlab !

Le PMP nous donne le contrôle optimal (cf. listing 5.1) $\bar{u}(z) \coloneqq p_v$ et l'on définit le hamiltonien maximisé (cf. listing 5.2) :

$$h(z) \coloneqq H(z, p^0, \bar{u}(z)) = p_x v + p_v(-\lambda v^2 + \bar{u}(z)) + \frac{1}{2} p^0 \, \bar{u}^2(z), \quad p^0 = -1 \text{ (cas normal)}.$$

```fortran
Subroutine control(t,n,z,iarc,npar,par,u)
    implicit none
    integer,                                 intent(in)   :: n,npar,iarc
    double precision,                        intent(in)   :: t
    double precision, dimension(2*n),        intent(in)   :: z
    double precision, dimension(npar),       intent(in)   :: par
    double precision,                        intent(out)  :: u

    u = z(n+2) ! u = pv

end subroutine control
```

Listing 5.1 – `afun.f90`

```fortran
Subroutine hfun(t,n,z,iarc,npar,par,h)
    implicit none
    integer,                                 intent(in)   :: n,npar,iarc
    double precision,                        intent(in)   :: t
    double precision, dimension(2*n),        intent(in)   :: z
    double precision, dimension(npar),       intent(in)   :: par
    double precision,                        intent(out)  :: h

    ! Local declarations
    double precision :: x,v,px,pv,lambda,u
```

```
    lambda   = par(7)
    x        = z(   1);    v    = z(   2)
    px       = z(n+1);     pv   = z(n+2)

    call control(t,n,z,iarc,npar,par,u)

    h = px*v + pv*(-lambda*v**2 + u) - 0.5d0*u**2

end subroutine hfun
```

Listing 5.2 – hfun.f90

5.2.1.2 Implémentation par l'utilisateur de la routine Fortran sfun

On définit la fonction de tir simple (cf. listing 5.3) :

$$
\begin{array}{rcl}
S\colon \mathbb{R}^2 & \longrightarrow & \mathbb{R}^2 \\
y & \longmapsto & S(y) := \Pi_q(\exp((t_f - t_0)\overrightarrow{h})(q_0, y)) - q_f
\end{array}
$$

```
Subroutine sfun(ny,y,npar,par,s)
    use mod_exphv4sfun
    implicit none
    integer,                                intent(in)  :: ny
    integer,                                intent(in)  :: npar
    double precision, dimension(ny),        intent(in)  :: y
    double precision, dimension(npar),      intent(in)  :: par
    double precision, dimension(ny),        intent(out) :: s

    !local variables
    double precision :: z0(4), zf(4), tspan(2)
    integer          :: n, iarc

    n          = 2                   ! Dimension of the state

    tspan(1)   = par(1)              ! t0
    tspan(2)   = par(2)              ! tf

    z0(   1)   = par(3)              ! x0
    z0(   2)   = par(4)              ! v0
    z0(n+1)    = y(1)                ! px0
    z0(n+2)    = y(2)                ! pv0

    iarc       = 1                   ! There is only one arc:
                                     !   compare with Bang-Bang case

    call exphv(tspan,n,z0,iarc,npar,par,zf)

    s(1)       = zf(1) - par(5)      ! x(tf) - xf
    s(2)       = zf(2) - par(6)      ! v(tf) - vf

end subroutine sfun
```

Listing 5.3 – Fonction de tir simple dans sfun.f90

5.2.1.3 Résolution avec HamPath de l'exemple

▷ **Exercice 5.1**. Résolution avec HamPath de l'exemple.

1. Récupérer sous Moodle le fichier `exemple_tir_simple_hampath.zip` et le dézipper (ce fichier est avec les sources du sujet 5).
2. Se rendre dans le répertoire `exemple_tir_simple_hampath/` et lancer la commande `hampath` (compilation).
3. Lancer MATLAB puis exécuter le script `main.m`.

5.2.2 Résolution avec HamPath d'un problème

5.2.2.1 Démarche

1. Créer le répertoire pour le problème et copier les fichiers de l'exemple dans ce répertoire ;
2. Modifier les routines FORTRAN décrivant le problème :
   - `hfun` dans `hfun.f90` ;
   - `control` dans `afun.f90` ;
   - `sfun` dans `sfun.f90`.
3. Lancer la commande `hampath` (compilation) ;
4. Modifier le fichier `main.m` ;
5. Lancer MATLAB puis exécuter le script `main.m`. Attention, le fichier `startup.m` crée lors de la compilation doit avoir été exécuté.

5.2.2.2 Fonctions Matlab disponibles

Fonction	Description
`hfun`, `hvfun`, `dhvfun`	hamiltonien maximisé et ses dérivées
`exphvfun`, `expdhvfun`	intégration numérique de `hvfun` et `dhvfun`
`sfun`, `sjac`	fonction de tir et sa jacobienne
`ssolve`	calcule un zéro de la fonction de tir
`hampath`	calcule des zéros de la fonction de tir en fonction d'un paramètre
`hampathset`	initialise les options (tolérances pour l'intégration numérique...)
`hampathget`	récupère les options

TABLE 5.1 – Description des fonctions disponibles sous MATLAB après compilation du problème via la commande `hampath`. On a accès à l'aide depuis MATLAB : » `help ssolve`.

5.2.2.3 Exercice

▷ **Exercice 5.2**. Retrouver les résultats des figures 3.1 et 3.2 avec HamPath.

5.3 Tir multiple et HamPath

5.3.1 Exemple

Considérons l'exemple de tir multiple de la documentation (disponible sous Moodle) de HamPath :

$$(\mathrm{P6}) \qquad \left\{ \begin{array}{l} J(t_f, u(\cdot)) = t_f \longrightarrow \min \\ \\ \dot{x}(t) = v(t), \\ \dot{v}(t) = -\lambda v(t)^2 + u(t), \quad u(t) \in [-1\,,1], \quad t \in [t_0\,,t_f] \text{ p.p.}, \\ \\ x(t_0) = x_0, \quad x(t_f) = x_f, \\ v(t_0) = v_0, \quad v(t_f) = v_f, \end{array} \right.$$

avec $t_0 = 0$, $q_0 \coloneqq (x_0, v_0) = (-1, 0)$ et $q_f \coloneqq (x_f, v_f) = (0, 0)$. On définit le vecteur de paramètres $par \coloneqq (t_0, x_0, v_0, x_f, v_f, \lambda)$.

5.3.1.1 Implémentation par l'utilisateur de la routine Fortran hfun

Le pseudo-hamiltonien est $H(q, p, u) = p_x v + p_v(-\lambda v^2 + u)$, avec $q \coloneqq (x, v)$ et $p \coloneqq (p_x, p_v)$. Le contrôle optimal est donné par :

$$\bar{u}(z) = \left\{ \begin{array}{ccc} +1 & \text{si} & p_v > 0 \\ u_s(z) & \text{si} & p_v = 0 \\ -1 & \text{si} & p_v < 0 \end{array} \right.$$

où $z \coloneqq (q, p)$ et où $u_s(z) \in [-1\,,1]$ est le contrôle singulier. On a ainsi cet ensemble de hamiltoniens :

$$h(z) = \left\{ \begin{array}{lll} h_+(z) \coloneqq H(z, +1) & \text{quand} & p_v > 0 \\ h_s(z) \coloneqq H(z, u_s(z)) & \text{quand} & p_v = 0 \\ h_-(z) \coloneqq H(z, -1) & \text{quand} & p_v < 0 \end{array} \right. .$$

Le PMP nous donne la structure de la solution : Bang-Bang avec deux arcs associés à h_+ puis h_-. Le rôle de la variable `iarc` dans les routines `control` et `hfun` est de choisir le contrôle associé à l'arc courant, *i.e.* c'est l'indice de l'arc courant.

```
if(iarc.eq.1)then
  u = 1d0
else ! iarc = 2
  u = -1d0
end if
```

Contrôle dans `afun.f90`

```
lambda   = par(6)
x        = z(   1);  v        = z(   2)
px       = z(n+1);  pv       = z(n+2)

call control(t,n,z,iarc,npar,par,u)

h = px * v + pv * (-lambda*v**2 + u)
```

Hamiltonien dans `hfun.f90`

5.3.1.2 Implémentation par l'utilisateur de la routine Fortran sfun

On définit la fonction de tir multiple (cf. listing 5.4) :

$$
\begin{array}{rccc}
S_\lambda\colon & \mathbb{R}^8 & \longrightarrow & \mathbb{R}^8 \\
& y := \begin{bmatrix} t_1 \\ t_f \\ p_{x,0} \\ p_{v,0} \\ z_1 \end{bmatrix} & \longmapsto & S_\lambda(y) := \begin{bmatrix} x(t_f) - x_f \\ v(t_f) - v_f \\ z_1 - z(t_1) \\ p_v(t_1) \\ h_-(z(t_f)) - 1 \end{bmatrix}
\end{array}
$$

où $z(t_1) := \exp((t_1 - t_0)\overrightarrow{h_+})(z_0)$, $z(t_f) := \exp((t_f - t_1)\overrightarrow{h_-})(z_1)$ et $z_0 := (x_0, v_0, p_{x,0}, p_{v,0})$. On considère le cas normal, *i.e.* $p^0 = -1$.

```
n        = 2
t0       = par(1); t1 = y(1); tf = y(2)
z0(  1)  = par(2); z0(  2) = par(3)  ! x0, v0
z0(n+1)  = y(3)  ; z0(n+2) = y(4)    ! px0, pv0
z1       = y(5:8)

! Integration on the first arc: u = +1
iarc     = 1; tspan    = (/t0, t1/)
call exphv(tspan,n,z0,iarc,npar,par,expz0)

! Integration on the second arc: u = -1
iarc     = 2; tspan    = (/t1, tf/)
call exphv(tspan,n,z1,iarc,npar,par,expz1)

call hfun(tf,n,expz1,iarc,npar,par,hf)

s(1)     = expz1(1) - par(4) ! Final condition on xf
s(2)     = expz1(2) - par(5) ! Final condition on vf
s(3:6)   = z1 - expz0        ! Matching condition z1 = z(t1)
s(7)     = expz0(n+2)        ! Switching condition pv(t1) = 0
s(8)     = hf - 1d0          ! Final Hamiltonian condition
```

Listing 5.4 – Fonction de tir multiple dans `sfun.f90`

5.3.1.3 Résolution avec HamPath de l'exemple

▷ **Exercice 5.3**. Résolution avec `HamPath` de l'exemple.

1. Récupérer et dézipper le fichier `exemple_tir_multiple_hampath.zip`.
2. Se rendre dans le répertoire `exemple_tir_multiple_hampath/` et lancer la commande `hampath` (compilation).
3. Lancer MATLAB puis exécuter le script `main.m`.

5.3.2 Résolution avec HamPath d'un problème

▷ **Exercice 5.4**. Retrouver les résultats avec `HamPath` de l'exercice 4.1 questions 5 et 6.

Sujet 6
Transfert orbital à temps minimal

6.1 Introduction

Le système considéré est un satellite de masse fixé m libéré par une fusée dans le plan de l'équateur ; l'orbite initiale du satellite est une ellipse de forte excentricité, voir figure 6.1. L'objectif de ce travail est de réaliser le transfert en temps minimal de cette orbite elliptique à une orbite circulaire géostationnaire.

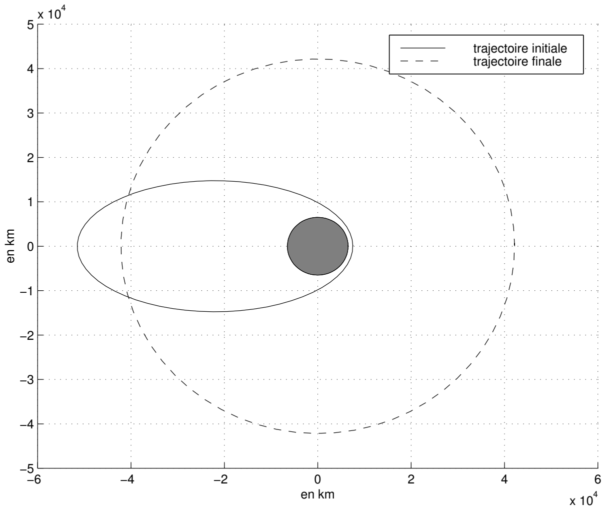

Figure 6.1 – Transfert orbital 2D.

6.2 Problème en temps minimal

Considérons le problème de transfert orbital à temps minimal suivant :

$$
(P7)\quad \begin{cases}
J(t_f, u(\cdot)) = t_f \longrightarrow \min \\
\dot{x}_1(t) = x_3(t), \\
\dot{x}_2(t) = x_4(t), \\
\dot{x}_3(t) = -\dfrac{\mu\, x_1(t)}{\|r(t)\|^3} + u_1(t), \\
\dot{x}_4(t) = -\dfrac{\mu\, x_2(t)}{\|r(t)\|^3} + u_2(t), \quad \|u(t)\| \leq \gamma_{\max}, \quad t \in [0\,, t_f] \text{ p.p.}, \quad u(t) := (u_1(t), u_2(t)), \\
x_1(0) = x_{0,1}, \quad x_2(0) = x_{0,2}, \quad x_3(0) = x_{0,3}, \quad x_4(0) = x_{0,4}, \\
r(t_f)^2 = r_f^2, \quad x_3(t_f) = -\sqrt{\dfrac{\mu}{r_f^3}}\, x_2(t_f), \quad x_4(t_f) = \sqrt{\dfrac{\mu}{r_f^3}}\, x_1(t_f),
\end{cases}
$$

avec $r(t) := \sqrt{x_1(t)^2 + x_2(t)^2}$. Les unités choisies sont le kilomètre pour les distances et l'heure pour les temps, *cf.* table 6.1.

Paramètre	Valeur	Unité
μ	$5.1658620912 \times 10^{12}$	km^3 h^{-2}
r_f	42165	km
$\gamma_{\max}$	388.8	km h^{-2}

TABLE 6.1 – Unité et valeurs des paramètres constants. Cette valeur de $\gamma_{\max}$ correspond à une accélération de 60N et à une masse de 2000kg, *cf.* $\gamma_{\max} = \frac{F_{\max}}{m} = \frac{60 \times 3600^2}{2000 \times 10^3} = 388.8$.

Question 8. Donner le pseudo-hamiltonien associé au problème (P7) (on se placera pour la suite dans le cas normal, *i.e.* $p^0 = -1$).

Question 9. Donner le contrôle optimal, *i.e.* la loi de commande maximisant le pseudo-hamiltonien.

Question 10. Déterminer la fonction de tir associé à (P7).

Remarque 6.1. Pour la résolution numérique, il sera préférable de remplacer l'équation $r(t_f)^2 = r_f^2$ par $r(t_f) = r_f$.

▷ **Exercice 6.1**. Calcul d'une BC-extrémale pour le problème (P7) avec `HamPath`.

1. Récupérer les sources du sujet 6. Vous pourrez utiliser le script `orbiteOf.m` pour l'affichage des trajectoires.
2. Résoudre la fonction de tir pour les conditions initiales :

$$x_{0,1} = -44000, \quad x_{0,2} = 0, \quad x_{0,3} = 0, \quad x_{0,4} = -10279.$$

   On note $y = (p_0, t_f)$ l'inconnue de la fonction de tir. On prendra comme point de départ pour la commande `ssolve` $y^{(0)} = (10^{-3}, 4 \times 10^{-4}, 10^{-3}, 10^{-4}, 4)$.

Sujet 7
Rendu des TP-Projet

Vous déposerez sous Moodle (la date limite est indiquée sur la page Moodle du cours) une archive contenant les codes et le rapport, sur la base des considérations données ci-après.

Les codes : Vous retournerez l'ensemble des codes, pour tous les sujets, en gardant la hiéarchie de base des répertoires. Les scripts devront pouvoir être exécutés sans modification de la part de l'enseignant afin de valider les exercices (seulement ceux présents dans le tableau 7.1).

Le rapport : Vous répondrez dans le rapport aux questions suivantes : 1 page 2 ; 2 page 5 ; 3 page 11 ; 4 page 16 ; 5 page 17 ; 6 page 17 ; 7 page 17 ; 8 page 25 ; 9 page 25 ; 10 page 25.

Vous complèterez dans votre rapport la colonne 3 (travail réalisé) du tableau 7.1. Vous mettrez **oui** si vous estimez que vous avez réalisé le travail et que l'enseignant peut le valider à l'aide du script associé, sans aucune modification de sa part. Sinon, vous mettrez **non**. Si par exemple, vous estimez pour la question 6.1 avoir fait les figures mais que votre solution n'est pas correcte, alors vous mettrez non dans la colonne 3.

Exercice	Rendu attendu	Travail réalisé (oui/non)
1.5	figure 1.2	
1.6	figure 1.3	
2.1	figure 2.1	
2.3 à 2.5	figure 2.2	
3.1	figures 3.1 et 3.2	
4.1	figure trajectoire et contrôle + affichage des sorties de `ssolve`	
4.2	figure trajectoire et contrôle + affichage des sorties de `ssolve`	
5.2	figures 3.1 et 3.2 (avec `HamPath`)	
5.4	figure trajectoire et contrôle + affichage des sorties de `ssolve` (avec `HamPath`)	
6.1	fig. trajectoire avec orbites initial et final, et contrôle + affichage des sorties de `ssolve` (avec `HamPath`)	

Table 7.1 – Table d'évaluation des TP. Vous rendrez dans le rapport ce tableau avec la colonne 3 remplie par vos soins. Si vous n'êtes pas sûr, vous pouvez mettre un “ ?”.

Evaluation : Vous serez évalué sur les réponses aux questions et sur les validations des tests.