Rapport de stage scientifique

Résolution numérique de problèmes d'optimisation dynamique en économie de l'environnement à l'aide du logiciel scientifique SCILAB

Stage proposé par le CIRED, réalisé avec le CEREVE, sous la direction de :
– M. Michel Cohen de Lara et M. Jean-Philippe Chancelier (CEREVE) ;
– M. Jean-Charles Hourcade (CIRED).

Sébastien Berthaud

Juillet 2000

Remerciements

Je tiens tout d'abord à remercier M. Jean-Philippe Chancelier, enseignant-chercheur au CEREVE, pour l'aide précieuse qu'il m'a apporté tout au long du stage, dès que je me voyais confronté à des problèmes techniques, ainsi que M. Pierre Carpentier, chercheur à l'ENSTA, qui m'a consacré de son temps pour me permettre de poser les bases des algorithmes utilisés utilisés par la suite.

Je remercie également M. Cohen de Lara pour son encadrement attentif durant le stage, et son aide salutaire quant à la rédaction de ce rapport.

Je voudrais aussi attirer l'attention sur le fait que M. Franck Lecocq, chercheur au CIRED, a lui aussi pris sur son temps, alors qu'il se trouvait en pleine période de soutenance de thèse, pour me fournir des informations importantes sur les modèles STARTS ainsi que des résultats de simulations numériques.

Je tiens aussi à remercier M. Jean-Charles Hourcade, M. Philippe Ambrosi, M. Vincent Gitz, M. Thierry Lepesant, Mme Yasmina Rakem ainsi que toutes les équipes du CIRED, et du CEREVE pour leur suivi tout au long du stage.

Résumé

Ce stage scientifique a pour but d'étudier et de simuler numériquement, avec le logiciel SCILAB, des modèles de comportement de pays, face à des contraintes environnementales sur les émissions de CO_2. Il se place dans le cadre d'une étude mise en place par le CIRED, financeur du stage.

Une description économique précise des modèles à utiliser a été réalisée au CIRED, par Franck Lecocq, pour aboutir aux modèles STARTS. Dans un premier temps, il a fallu mettre ces derniers sous une forme ¡¡mathématiquement correcte¿¿, c'est à dire sous une forme de *problème d'optimisation dynamique.*

Ensuite, des outils informatiques ont été développés pour permettre à SCILAB de traiter les problèmes posés. Ces outils ont été voulus très généraux, afin d'être réutilisables par le CIRED pour tous les futurs modèles économiques. Il s'agit em l'occurence des macros *dynoptim* et *dynoptimsc* qui permettent de traiter des problèmes d'optimisation dynamique.

Enfin, des comparaisons ont été réalisées entre les résultats fournis par le programme fonctionnant avec SCILAB et ceux donnés par un autre logiciel, GAMS, avec lequel Franck Lecocq avait commencé à simuler les modèles STARTS.

Mots-clés. Optimisation, système dynamique, algorithme d'Uzawa, dualité, Lagrangien augmenté, SCILAB, contrainte environnementale, coût-avantage, coûts-efficacité, effet de serre, optimisation dynamique.

Résumé

Table des figures

Liste des tableaux

Introduction

Le CEREVE, centre d'enseignement et de recherche sur l'eau, la ville et l'environnement, est issu de la fusion entre le CERGRENE et le LABAM. C'est à la fois un laboratoire de recherche pluridisciplinaire, un centre d'expertise et un support de formations.

Placé sous la tutelle de l'ENPC, de l'ENGREF et de l'UPVM, ses objectifs sont triples : faire progresser les connaissances scientifiques, participer à la mise en place d'instruments de la gestion intégrée en environnement, et contribuer à la formation des praticiens et chercheurs de demain.

Pour mener à bien ses recherches, le CEREVEcombine efficacement un savoir-faire dans la mise en œuvre de campagnes et d'enquêtes de terrain, des compétences analytiques et une maîtrise de la modélisation.

Le CIRED, pour Centre international de recherche sur l'environnement et le développement, a été créé en 1973 par le professeur à l'École des hautes études en sciences sociales, Ignacy Sachs. Le but premier de ce laboratoire est d'étudier les intéractions entre l'environnement, la gestion des ressources naturelles et le développement économique.

En 1979, le CIRED est devenu une unité de recherche du CNRS, et depuis 1987, M. Jean-Charles Hourcade en est le directeur.

Le CIRED est associé à de nombreux cours et partenariats, avec notamment l'EHESS, l'ENGREF, l'Université Paris X, et l'Université de Marne la Vallée ainsi que l'ENPC.

Le logiciel SCILAB est un logiciel mathématique gratuit, librement distribué, créé à l'initiative de l'INRIA. M Jean Philippe Chancelier, qui travaille actuellement au CEREVE, fait partie de l'équipe des développeurs de ce logiciel.

SCILAB est très évolutif, chacun peut y rajouter ses travaux sous formes de ¡¡contributions¿¿.

De plus, il est doté d'une interface très facile d'utilisation, permettant un apprentissage aisé et l'écriture de programmes très lisibles.

SCILAB est utilisé au CEREVE, mais aussi par les élèves de l'École nationale des ponts et chaussées.

Objectifs et encadrements du stage

Les problèmes d'optimisation dynamique, *i.e.* des problèmes d'optimisation où le temps apparaît, sont très nombreux en économie, et le logiciel GAMS se prête relativement bien à la résolution de ces problèmes, tout du moins en ce qui concerne les petits modèles.

Cependant ce logiciel n'est pas très pratique d'utilisation, il nécessite l'écriture et la lecture de multiples fichiers à chaque utilisation. C'est pourquoi le CIRED cherchait un logiciel plus souple, afin de pouvoir traiter de manière plus aisée des problèmes de plus en plus complexes.

M Michel Cohen de Lara proposa donc au CIRED de se tourner vers le logiciel SCILAB, afin de profiter de sa grande souplesse d'utilisation. En revanche, comme le logiciel SCILAB ne posséde pas d'outil pouvant être appliqué à la résolution de problèmes d'optimisation dynamique, il y a un enjeu à en créer.

Les objectifs de ce stage sont

- d'tiliser SCILAB pour mettre en place les modélisations informatiques des problèmes économiques du CIRED ;
- de développer des outils SCILAB pour l'optimisation dynamique, en vue de la résolution numérique de problèmes d'économie de l'environnement.

L'encadrement de ce stage est assuré par :

- M. Michel Cohen de Lara, enseignant-chercheur au CEREVE, tuteur du stage ;
- M. Jean-Philippe Chancelier, enseignant-chercheur au CEREVE ;
- M. Jean-Charles Hourcade, directeur du CIRED, responsable du stage.

Les modèles à étudier

Les modèles STARTS (pour *Sectoral Trajectories with Adaptation and Response Turnover of Stocks*) constituent une famille de modèles d'optimisation des politiques de réduction des émissions de gaz à effet de serre développée au CIRED par Franck Lecocq et Jean-Charles Hourcade avec l'aide technique de Naceur Ben Chaabane et Sylvain Cori.

Ces modèles vont permettre de définir de ce que SCILAB devra être capable de résoudre. De plus, certains modèles STARTS ont déjà été programmés avec GAMS ; ils seront donc utiles pour vérifier le bon fonctionnement des programmes écrits en SCILAB.

Il existe aujourd'hui huit versions des modèles STARTS qu'il est possible de regrouper en deux catégories

- Optimisation partielle ou optimisation globale : dans le premier cas, nous ne disposons comme variable de contrôle que du niveau de réduction des émissions (ou abattement) ; dans le second, nous contrôlons l'ensemble de l'économie, et en particulier les décisions d'épargne des agents, et nous rajoutons donc une nouvelle variable de commande (la consommation) et une nouvelle variable d'état (le capital).
- Coûts-efficacité ou coûts-bénéfices : dans le premier cas, il s'agit de minimiser les coûts (ou de maximiser l'utilité) sous contrainte de non dépassement d'une contrainte environnementale ; dans le second, il existe un arbitrage explicite entre les coûts de l'abat-

tement et les coûts des impacts ; si la seconde manière est plus rationnelle pour un économiste, elle oblige par contre à aborder la question de la valorisation monétaire des impacts du changement climatique.

La démarche

Le stage est divisé en quatre parties

- Mise en forme mathématique des problèmes économiques à étudier (problème d'optimisation dynamique).
- Développement d'une primitive générale d'optimisation dynamique sous SCILAB.
- Programmation sous SCILAB des problèmes STARTS.
- Résolution numérique des problèmes STARTS.

Notations utilisées

Notations mathématiques générales

$\mathcal{F}(E, H)$	l'ensemble des fonctions de E dans H ;
$C^0(E, H)$	l'ensemble des fonctions continues de E dans H ;
$C^n(E, H)$	l'ensemble des fonctions de E dans H dérivables n fois, et telles que leurs n premières dérivées soient continues sur E ;
∇J	le gradient de J (on suppose $J \in C^0(E, H)$, différentiable) ;
f'	la dérivée de la fonction f ; (cependant, dans certains cas, qui seront signalés, il faudra lire M' comme étant la transposée de la matrice M et non la dérivée de la fonction M).

Dans les problèmes de commande optimal, u et X désigneront respectivement l'état et la commande du système. L'état à un instant donné sera supposé de taille n, et la commande de taille p.

Si a désigne une suite d'éléments de $\mathbb{R}^n$, alors a_t est le t^{e} terme de la suite.

Notations propres aux modèles STARTS

$M_{t\in\mathbb{N}} \in \mathbb{R}$	les concentrations en CO_2 ;
$a_{t\in\mathbb{N}} \in \mathbb{R}$	l'abattement des émissions de CO_2 ;
$\overline{E}_{t\in\mathbb{N}} \in \mathbb{R}$	les émissions de gaz à effet de serre de référence ;
$\kappa_{t\in\mathbb{N}} \in \mathbb{R}$	le PIB.

Chapitre 1

Description mathématique des modèles STARTS

Cette partie à pour objet de faire un tour d'horizon des modèles STARTS existants, en mettant l'accent sur la clarification de leur structure mathématique. Nous avons gardé ici les dénominations données aux modèles par le CIRED.

1.1 Optimisation partielle sous contrainte environnementale

Le problème posé consiste à modéliser un pays qui doit mener une politique de réduction des émissions de gaz à effet de serre en deçà d'un seuil préalablement fixé, tout en minimisant les coûts de cette politique.

1.1.1 Modèle en temps continu

Nous supposons que les émissions de référence $\overline{E_t}$, c'est à dire l'évolution des émissions si aucune mesure n'est prise, sont connues. On dispose de moyens pour réduire à l'instant t les émissions d'une fraction $a_t E_t$: a_t sera appelé *abattement*. On note M_t l'émission réelle de gaz à effet de serre à l'instant t.

L'objectif consiste à minimiser la somme actualisée (à un taux $\rho > 0$) des coûts de réduction des émissions de gaz à effet de serre

$$J(a) = \int_0^{+\infty} C(a(t), a'(t), t) e^{-\rho t} dt, \tag{1.1}$$

sous la contrainte environnementale de non-dépassement d'un seuil préalablement fixé

$$M(t) \leq \overline{M}. \tag{1.2}$$

Le coût d'abattement s'écrit

$$C(a(t)t, a'(t), t) = \alpha \overline{E(t)} a(t)^{\nu} \lambda(t) \gamma(a'(t)), \tag{1.3}$$

et il est composé de trois facteurs

– un terme dépendant de $a(t)$ en $\alpha \overline{E(t)} a(t)^{\nu}$ qui est le coût d'abattement ¡¡brut¿¿ ;

– un terme de progrès technique $\lambda(t)$, où λ est une fonction décroissante vérifiant $\lambda(0) = 1$ et $\lim_{t\to\infty} \lambda(t) = \overline{\lambda}$ $(0 < \overline{\lambda} < 1)$;
– un terme $\gamma(a'(t))$ qui traduit l'inertie du capital, par exemple de la forme

$$\gamma(a) = \begin{cases} 1 & \text{si } a < \overline{\gamma} \\ \frac{a}{\overline{\gamma}} & \text{sinon.} \end{cases}$$

Pour modéliser l' évolution des concentrations en CO_2, nous utilisons la dynamique d'accumulation du CO_2 suivante :

$$\begin{cases} M(0) &= M_0 \\ M'(t) &= \beta\overline{E(t)}(1 - a(t)) - \sigma(M(t) - M_{-\infty}) \end{cases}. \tag{1.4}$$

Ici, β représente une fraction $(0 < \beta < 1)$ des émissions de CO_2 stockée dans l'atmosphère, et σ une fraction $(0 < \sigma < 1)$ du CO_2 excédentaire pré-industriel $M_{-\infty}$ réabsorbée par le sol et les océans. Nous supposons que la concentration initiale M_0 en CO_2 est supérieure à la concentration pré-industrielle : $M_0 > M_{-\infty}$.

Les émissions de référence $\overline{E(t)}$ sont croissantes (éventuellement à une vitesse elle aussi croissante au départ) avant de décroître puis de tendre vers 0 quand t temps vers l'infini.

Ici, $a'(t)$ est la variable de *commande* du problème, alors que $a(t)$ et $M(t)$ sont des variables *d'état* du système. En effet, il suffit de connaître le couple (a_0, M_0) ainsi que l'ensemble des commandes $a'(t)$ pour décrire complètement le système *via* la dynamique (1.4).

1.1.2 Modèle en temps discret

Du modèle précédent nous pouvons déduire un modèle en temps discret en transformant la quantité $a'(t)$ en $\left(\frac{a_t - a_{t-1}}{\Delta t}\right)$ où Δt est un pas de temps adapté ; la variable a_t est devenue variable de *commande*.

Bien entendu, le modèle obtenu alors est très proche du précédent. En ce qui concerne le coût, une somme discrète vient remplacer l'intégrale, de sorte que le critère à minimiser devient

$$J(a) = \sum_{t=0}^{T-1} C(a_t, a_{t-1}, t)\frac{1}{(1+\rho)^t}, \tag{1.5}$$

avec

$$C(a_t, a_{t-1}, t) = \alpha\overline{E_t}{a_t}^{\nu}\lambda(t)\gamma(a_t, a_{t-1}). \tag{1.6}$$

La dynamique devient :

$$\begin{cases} M_{t_{|t=0}} &= M_0 \\ M_t &= M_{t-1} + \Delta t\Big(\beta\overline{E}_{t-1}(1 - a_{t-1}) - \sigma(M_{t-1} - M_{-\infty})\Big). \end{cases} \tag{1.7}$$

Le modèle obtenu alors est une discrétisation possible du modèle en temps continu présenté plus haut, et il faut noter quelques petites modifications : dans le premier cas, la variable de *commande* était le vecteur $a'(t)$ et *l'état* était $\begin{pmatrix} a(t) \\ M(t) \end{pmatrix}$, alors que dans le second cas, la variable de *commande* est a_t et *l'état* devient le vecteur $x_t = \begin{pmatrix} a_{t-1} \\ M_t \end{pmatrix}$.

Dans le premier cas, si nous connaissons $a'(t)$ pour $t < \theta$, ainsi que $a(0)$ et $M(0)$, alors nous pouvons obtenir $a(\theta)$ ainsi que $M(\theta)$. Pour cela, il suffit d'intégrer

- $a'(t)$ pour retrouver $a(\theta) = \int_0^\theta a'(t)dt + a(0)$,
- puis l'équation différentielle $\dot{M(t)} = \beta\overline{E(t)}(1 - a(t)) - \sigma(M(t) - M_{-\infty})$.

Dans le second cas, si nous connaissons a_t pour $t \in \{0, \ldots, \theta\}$, ainsi que a_0 et M_0, alors nous pouvons obtenir a_θ ainsi que M_θ. Pour cela, il suffit de partir de M_0 de calculer M_t pour $t \in \{0, \ldots, \theta\}$ *via* la dynamique (1.7)pour enfin obtenir M_θ.

1.2 Optimisation partielle en analyse coûts-avantages

Le problème posé consiste maintenant à modéliser un pays qui doit mener une politique de réduction des émissions de gaz à effet de serre, en minimisant à la fois, les coûts

- des impacts futurs des changements climatiques ;
- de la politique de réduction des émissions.

1.2.1 Modèle en temps continu

Ce modèle est très proche de ceux étudiés précédemment, car les variables *de commande* et *d'état* restent identiques, ainsi que la dynamique. Cependant, la contrainte environnementale (1.2) a disparu, et maintenant le coût des impacts des dommages dus à la pollution est pris en compte dans la fonction *objectif*.

Ainsi, la dynamique reste inchangée, à savoir

$$\left\{ \begin{array}{lcl} M(0) & = & M_0 \\ M'(t) & = & \beta\overline{E(t)}(1 - a(t)) - \sigma(M(t) - M_{-\infty}), \end{array} \right. \tag{1.8}$$

et le critère à minimiser devient

$$J(a, M) = \int_0^{+\infty} \Big(C(a(t), a'(t), t) + d(M(t), t) \Big) e^{-\rho t} dt. \tag{1.9}$$

Le terme $C(a(t), a'(t), t)$ du coût de l'abattement dans l'expression du critère est identique à l'expression (1.3), et le terme correspondant au coût des dommages s'écrit comme le produit du PIB $\kappa(t)$ de référence et d'un indicateur de dommage $\xi(t)$ variant entre 0 et 1, soit :

$$d(M(t), t) = \kappa(t)\xi(M(t)). \tag{1.10}$$

On impose simplement à $\kappa(t)$ et $\xi(t)$ d'être croissantes.

1.2.2 Modèle en temps discret

De la même manière que pour le modèle d'optimisation partielle sous contrainte environnementale, nous allons définir un modèle en temps discret en remplaçant encore $a'(t)$ par $\Big(\frac{a_t - a_{t-1}}{\Delta t}\Big)$.

Nous sommes donc amenés à considèrer le problème suivant :

On minimise le critère

$$J(a, M) = \sum_{t=0}^{T-1} \Big(C(a_t, a_{t-1}, t) + d(M_t, t)\Big) \frac{1}{(1+\rho)^t}. \tag{1.11}$$

Les états sont liés par la dynamique (1.7), à savoir :

$$\begin{cases} M_{t_{|t=0}} &= M_0 \\ M_t &= M_{t-1} + \Delta t\Big(\beta\overline{E_{t-1}}(1 - a_{t-1}) - \sigma(M_{t-1} - M_{-\infty})\Big), \end{cases}$$

et les hypothèses sur la fonction de dommage restent les mêmes qu'au paragraphe (1.2.1).

1.3 Synthèse

Tous les modèles présentés ci-dessus – aussi bien en temps continu, qu'en temps discret – ne sont que des cas particuliers de problèmes plus généraux et classiques en commande optimale. En effet, tous les problèmes étudiés ci-dessus peuvent se mettre sous la forme générale suivante (Culioli, 1994) :

En temps continu

Soit $T > 0$ (éventuellement $T = +\infty$), on minimise en $x \in C^0\Big([0,T];\mathbb{R}^n\Big)$ et en $u \in C^0\Big([0,T];\mathbb{R}^p\Big)$ le critère

$$J(x,u) = J_1(x,u) + J_2(x,u) \quad \text{avec,} \tag{1.12}$$

$$\text{le coût intégral} \qquad J_1(x,u) = \int_0^T l(x(t), u(t), t)dt \ , \tag{1.13}$$

$$\text{le coût final} \qquad J_2(x,u) = \Phi(x(T), T) \ , \tag{1.14}$$

où l'état x et la commande u sont liés par la dynamique

$$x' = F(x(t), u(t), t), \quad t \in [0,T] \ , \tag{1.15}$$

avec des conditions aux limites (éventuellement vides)

$$h_0(x(0),0)=0, \qquad h_T(x(T),T)=0 \tag{1.16}$$

et des contraintes pouvant prendre des formes diverses ($\forall t \in [0,T]$)

$$\text{les contraintes d'égalité} \quad k_1(x(t),u(t),t)=0, \tag{1.17}$$
$$\text{les contraintes d'inégalité} \quad k_2(x(t),u(t),t)\leq 0, \tag{1.18}$$
$$\text{le domaine d'admissibilité} \quad u(t)\in U(t), \tag{1.19}$$

Il est à noter que l'on distingue traditionnellement les contraintes d'état, qui sont une forme particulière des contraintes instantanées (1.17), et qui s'écrivent :

$$K_1(x(t),t)=0, \text{ ou } K_2(x(t),t)\leq 0 \ .$$

En temps discret

Soit $T>0$ (éventuellement $T=+\infty$), on minimise en $x=(x_0,\ldots,x_T)\in\mathbb{R}^{n(T+1)}$ ainsi qu'en $u=(u_0,\ldots,u_{T-1})\in\mathbb{R}^{pT}$, le critère

$$J(x,u) = J_1(x,u)+J_2(x,u) \text{ avec,} \tag{1.20}$$
$$\text{le coût intégral} \quad J_1(x,u)=\sum_0^{T-1} l(x_t,u_t,t)dt \ , \tag{1.21}$$
$$\text{le coût final} \quad J_2(x,u)=\Phi(x_T,1) \ , \tag{1.22}$$

où l'état x et la commande u sont liés par la dynamique

$$x_{t+1}=F(x_t,u_t,t), \quad t\in\{0,\ldots,T-1\} \ , \tag{1.23}$$

avec des conditions aux limites (éventuellement vides)

$$h_0(x_0,0)=0, \qquad h_T(x_T,T)=0 \tag{1.24}$$

et des contraintes pouvant prendre des formes diverses ($\forall t\in\{0,\ldots,T-1\}$)

$$\text{les contraintes d'égalité} \quad k_1(x_t,u_t,t)=0, \tag{1.25}$$
$$\text{les contraintes d'inégalité} \quad k_2(x_t,u_t,t)\leq 0 \ , \tag{1.26}$$
$$\text{le domaine d'admissibilité} \quad u_t\in U_t. \tag{1.27}$$

Toutes les fonctions intervenant dans les contraintes ainsi que dans le critère sont au moins dérivables par rapport à tous leurs arguments, et la borne T peut être laissée libre et sera alors considérée comme une commande.

Il est à noter que pour le modèle en temps discret, la commande u est définie sur $0, \ldots, T$ et l'état sur $0, \ldots, T+1$, alors qu'en ce qui concerne le modèle en temps continu, la commande u et l'état x sont tous deux définis sur $[0, T]$.

Lorsque le critère comporte à la fois un coût intégral (1.13), et un coût final (1.14), on parle de problème de *Bolza*. Si le critère ne comporte que le terme de coût intégral, on parle de problème de *Lagrange*, et si le critère ne comporte que le terme de coût final, on parle de problème de *Meyer* (Culioli, 1994).

1.4 Retour sur les modèles d'optimisation STARTS

En ce qui concerne les modèles STARTS exposés précédemment, nous avons vu que pour les modèles en temps continu, la commande était $a'(t)$, l'état était $\begin{pmatrix} a(t) \\ M(t) \end{pmatrix}$, que dans le cadre des modèles en temps discret, la variable de commande est a_t et l'état $x_t = \begin{pmatrix} a_{t-1} \\ M_t \end{pmatrix}$.

Nous allons maintenant étudier plus particulièrement les deux modèles STARTSen temps discret, et ces deux modèles peuvent effectivement se présenter sous la forme d'un problème de Bolza.

Le *modèle d'optimisation partielle en analyse coûts-avantages* se présente sous la forme d' un problème dynamique de *Bolza* en horizon fini sans contrainte instantanée.

Nous devons minimiser en $a \in \mathbb{R}^T$ et en $x \in \mathbb{R}^{2(T+1)}$, le critère

$$J(a, x) = \sum_{t=0}^{T-1} \left(C(a_t, x_t^{(1)}, t) + d(x_t^{(2)}, t) \right) \frac{1}{(1+\rho)^t}, \tag{1.28}$$

où

$$C(a_t, x_t^{(1)}, t) = \alpha \overline{E_t} {a_t}^{\nu} \lambda(t) \gamma(a_t, x_t^{(1)}). \tag{1.29}$$

et d est la fonction dommage définie en (1.11).

Les états $x_t = \begin{pmatrix} x_T^{(1)} \\ x_T^{(2)} \end{pmatrix}$ sont liés par la dynamique

$$\left\{ \begin{array}{rcl} x_0 & = & \begin{pmatrix} a_0 \\ M_0 \end{pmatrix} \\ x_t & = & \begin{pmatrix} 1 \\ -\Delta t \beta \overline{E}_{t-1} \end{pmatrix} a_{t-1} + \begin{pmatrix} 0 & 0 \\ 0 & 1 - \Delta t \sigma \end{pmatrix} x_{t-1} + \begin{pmatrix} 0 \\ \Delta t(\beta \overline{E}_{t-1} + \sigma M_{-\infty}) \end{pmatrix} \end{array} \right. \tag{1.30}$$

Le *modèle d'optimisation partielle sous contrainte environnementale* est lui aussi un problème dynamique de *Bolza* en horizon fini, mais contrairement au modèle précédent, on y a introduit des contraintes d'état.

Le critère à minimiser se présente sous la même forme qu'en (1.28) mais sans le terme d correspondant au coût des dommages

$$J(a, x) = \sum_{t=0}^{T-1} C(a_t, x_t^{(1)}, t) \frac{1}{(1+\rho)^t}, \tag{1.31}$$

où

$$C(a_t, x_t^{(1)}, t) = \alpha \overline{E_t} {a_t}^{\nu} \lambda(t) \gamma(a_t, x_t^{(1)}). \tag{1.32}$$

Les états sont toujours liés par la dynamique (1.30), et ils doivent vérifier la contrainte d'état supplémentaire

$$\begin{array}{c} 0 \leq x_T^{(1)} \leq 1 \\ 0 \leq x_T^{(2)} \leq \overline{M} \end{array} \tag{1.33}$$

Cependant, dans aucun des deux modèles, nous n'avons exprimé de coût final, si bien que dans le cas du problème d'optimisation en coûts-avantages, nous ne prenons pas en compte le coût des dommages provoqués par les émissions M_{T+1} : ainsi la commande a_T sera toujours nulle, ce qui peut parfois légerement fausser les trajectoires optimales même pour $t << T$. Pour palier à cela, nous pouvons introduire un coût final

$$\Phi(x(T), T) = d(x_T^{(2)}, T),$$

où d est la fonction de dommages utilisée sous le signe somme (1.11).

Modèle STARTS **d'optimisation sous contraintes environnementales**

On cherche

$$\min_{\substack{u_0,\dots,u_{T-1}\in\mathbb{R}\\ x_0,\dots,x_T\in\mathbb{R}^2}} J(u,x),$$

$$\text{avec}\quad J(u,x)=\sum_{t=0}^{T-1}\frac{\alpha\overline{E_t}{u_t}^{\nu}\lambda(t)\gamma(u_t,x_t^{(1)})}{(1+\rho)^t},$$

en tenant compte de la dynamique

$$\begin{pmatrix} x_t^{(1)}\\ x_t^{(2)}\end{pmatrix}=\begin{pmatrix} u_{t-1}\\ x_{t-1}^{(2)}+\Delta t\Big(\beta\overline{E}_{t-1}(1-u_{t-1})-\sigma(x_{t-1}^{(2)}-M_{-\infty})\Big)\end{pmatrix}$$

et de la contrainte d'état

$$\begin{aligned} 0\le x_T^{(1)}&\le 1\\ 0\le x_T^{(2)}&\le \overline{M}\end{aligned}$$

Modèle STARTS **d'optimisation en coûts-avantages**

On cherche

$$\min_{\substack{u_0,\dots,u_{T-1}\in\mathbb{R}\\ x_0,\dots,x_T\in\mathbb{R}^2}} J(u,x),$$

$$\text{avec}\quad J(u,x)=\sum_{t=0}^{T-1}\left(\frac{\alpha\overline{E_t}{u_t}^{\nu}\lambda(t)\gamma(u_t,x_t^{(1)})+\kappa(t)\xi(x_t^{(2)})}{(1+\rho)^t}\right)+\frac{\kappa(T)\xi(x_T^{(2)})}{(1+\rho)^T}$$

en tenant compte de la dynamique

$$x_t=\begin{pmatrix} u_{t-1}\\ x_{t-1}^{(2)}+\Delta t\Big(\beta\overline{E}_{t-1}(1-u_{t-1})-\sigma(x_{t-1}^{(2)}-M_{-\infty})\Big)\end{pmatrix}.$$

Chapitre 2

Développement d'une primitive SCILAB pour la résolution du problème de Bolza sans contrainte

Nous allons tout d'abord traiter le cas du problème de *Bolza*, (*i.e.* avec un critère comportant à la fois un coût intégral et un coût final), sans contrainte, ce qui nous permettra de résoudre des problèmes du type *optimisation partielle en analyse coûts-avantages*.

2.1 Mise en forme algorithmique du problème de Bolza

2.1.1 Le problème de Bolza

Nous allons nous intéresser à la résolution du problème de Bolza sans contrainte, sans condition aux limites, et avec des conditions d'admissibilité simples, *i.e* aux problèmes de la forme suivante :

Soit $T > 0$ (éventuellement $T = +\infty$), on minimise en $x_0, \ldots, x_T \in \mathbb{R}^n$,
et en $u_0, \ldots, u_{T-1} \in \mathbb{R}^p$, le critère

$$J(x,u) = J_1(x,u) + J_2(x,u) \text{ avec,} \tag{2.1}$$

$$\text{le coût intégral} \qquad J_1(x,u) = \sum_0^{T-1} l(x_t, u_t, t) \tag{2.2}$$

$$\text{le coût final} \qquad J_2(x,u) = \Phi(x_T, T)\ , \tag{2.3}$$

en tenant compte de la dynamique

$$x_{t+1} = F(x_t, u_t, t), \quad t \in \{0, \ldots, T-1\}, \tag{2.4}$$

et éventuellement d'un domaine d'admissibilité

$$u_t \in \mathcal{U} \text{ pour tout } t \in \{0, \ldots, T-1\}. \tag{2.5}$$

2.1.2 Transformation du problème dynamique de Bolza à horizon fini en un problème d'optimisation statique

L'idée de la transformation est la suivante : x_0 étant fixé, il suffit de connaître u_t pour $t \in \{0, \ldots, T-1\}$ pour pouvoir évaluer tous les x_t. Nous pouvons donc voir le critère à minimiser $J(x, u)$ comme une fonction de u seulement.

Il ne reste alors qu'à minimiser ce critère en u pour obtenir la commande optimale, ce qui pourra se faire avec un algorithme de recherche de minimum classique (programmé sous SCILAB par exemple).

Il nous faut montrer ici, qu'à x_0 fixé, il existe une fonction Ψ_{x_0} de $\mathbb{R}^{(p \cdot T)}$ dans $\mathbb{R}^{n \cdot (T+1)}$ telle que

$$x_t = \Psi_{x_0}(u).$$

En effet, il suffit de considérer la suite de fonctions

$$\begin{array}{rcl} \psi_0(u) & = & F(u_0, x_0, 0) \ , \\ \psi_1(u) & = & F(u_1, F(u_0, x_0, 0), 1) \ , \\ \vdots & & \vdots \\ \psi_T(u) & = & F(u_{t-1}, F(u_{t-2}, F(\ldots F(u_0, x_0, 0) \ldots), t-2)t-1) \ , \end{array} \tag{2.6}$$

et de poser

$$\Psi_{x_0}(u) = \begin{pmatrix} \psi_0(u) \\ \psi_1(u) \\ \vdots \\ \psi_T(u) \end{pmatrix} \in \mathbb{R}^{n \cdot (T+1)} \tag{2.7}$$

afin d'obtenir

$$x = \begin{pmatrix} x_0 \\ \vdots \\ x_T \end{pmatrix} = \Psi_{x_0}(u) \ . \tag{2.8}$$

Nous pouvons donc définir

$$\mathcal{J}(u) = J(u, \Psi_{x_0}(u)). \tag{2.9}$$

La fonction $\mathcal{J}(u)$ ainsi définie sur $\mathbb{R}^{pT}$ est continue, dérivable à dérivée continue. En effet, la fonction F est C^1 par rapport à toutes ses variables, donc par composition, la fonction Ψ_{x_0} est aussi C^1, tout comme, pour les mêmes raisons, la fonction $\mathcal{J}(u)$.

Le problème de Bolza peut donc maintenant être traité de manière équivalente comme un problème d'optimisation statique en u. Dorénavant $\mathcal{J}$ désignera aussi bien le critère sous sa forme statique (2.9), que sous sa forme dynamique (2.1).

2.1.3 L'algorithme de calcul de l'état adjoint

Nous allons tenter maintenant de résoudre numériquement le problème d'optimisation statique

$$\min_{u \in \mathcal{U}} \mathcal{J}(u) \tag{2.10}$$

avec

- x_0 fixé ;
- $\mathcal{J}(u) = J(u, \Psi_{x_0}(u))$;
- $J(u, x) = \sum_{t=0}^{T-1} l(u_t, x_t, t) + \Phi(x_T, T)$.

Nous avons vu que la fonction $\mathcal{J}$ était C^1, ce qui est une condition suffisante pour pouvoir rechercher des minima locaux, mais le gradient ne peut pas être exprimé simplement.

Nous pourrions utiliser le calcul de dérivée des fonctions composées

$$\frac{d}{du}(f \circ g(u)) = f'(g(u)) \circ g'(u),$$

mais vue la forme de la fonction $\mathcal{J}$, à savoir

$$\mathcal{J}(u) = J(u, \Psi_{x_0}(u)),$$

cette manière de calculer le gradient risque d'être vraiment coûteuse en temps.

En effet, conformément à l'expression (2.7), calculer le gradient de Ψ_{x_0} nécessite le calcul de gradient de T fonctions qui sont encore des fonctions composées. Ainsi par exemple, la dernière composante de la fonction Ψ_{x_0}, c'est à dire la fonction ψ_T définie en (2.6), est composée de T fois la fonction dynamique F.

Cependant il existe une autre méthode pour permettre des calculs de gradients beaucoup plus rapides ; pour cela il faut revenir à la forme dynamique du critère

$$J(u,x,t) = \sum_{t=0}^{T-1} l(u_t, x_t, t) + \Phi(x_T, T), \tag{2.11}$$

avec

$$\left\{ \begin{array}{rcl} x_{t_{|t=0}} & = & x_0 \\ x_{t+1} & = & F(u_t, x_t, t), \end{array} \right.$$

et il faut introduire une nouvelle variable, appelée état adjoint, de même dimension que l'état.

Cette nouvelle variable Λ, propre à chaque type de problèmes, permet, grâce à un algorithme simple, le calcul de gradients non explicites.

Ici nous voulons calculer le gradient *en u* d'une fonction de la forme (2.11), et nous allons donc introduire l'état adjoint Λ vérifiant l'équation récurrente rétrograde

$$\left\{ \begin{array}{rcl} \forall t & \in & \{T-1, \ldots, 0\} \\ \Lambda_T & = & (\frac{d\Phi}{dx}(x_T))', \\ \Lambda_t & = & (\frac{\partial F}{\partial x}(u_{t+1}, x_{t+1}, t+1))' \Lambda_{t+1} + (\frac{\partial l}{\partial x}(u_{t+1}, x_{t+1}, t+1))'. \end{array} \right. \tag{2.12}$$

Le gradient de la fonction $\mathcal{J}$ s'écrit alors

$$\nabla \mathcal{J} = \sum_{t=0}^{T-1} \Lambda_t' \frac{dF}{du}(u_t, x_t, t) + (\frac{dl}{du}(u_t, x_t, t))' L_t, \tag{2.13}$$

où L_t est la matrice ligne dont tous les éléments sont nuls, sauf le t^{e} qui vaut 1.

Les calculs sont alors beaucoup plus simples et surtout moins nombreux, donc le calcul du gradient s'effectue beaucoup plus rapidement. En effet, pour le calcul d'un gradient en un point u donné, le nombre d'évaluations numériques des fonctions données (*i.e.* le nombre d'évaluation des fonctions l, F, et Φ ainsi que de leur dérivées respectives par rapport à la commande et à l'état) est en $O(T^2)$ avec la méthode de dérivation des fonctions composées, alors qu'en utilisant l'état adjoint ce même nombre est en $O(T)$.

2.2 La création de la macro SCILAB *dynoptim*

Par la suite, le terme
- *primitive* désignera une fonction programmée en C où en FORTRAN, mais appelable depuis SCILAB ;
- *macro* SCILAB désignera une fonction programmée en langage SCILAB.

Présentation *d'optim*

Le logiciel SCILAB dispose d'un outil adapté à la résolution de problème d'optimisation statique. La fonction *optim* est un outil puissant pour procéder à des recherches de minimum sans contrainte de fonction différentiable, utilisant au choix (Culioli, 1994)

– l'algorithme de gradient conjugué ;
– l'algorithme du quasi Newton.

Il suffit de préciser à cette fonction l'algorithme à utiliser, la fonction à minimiser, et le point de départ de l'algorithme, (*i.e.* la valeur de la commande pour laquelle l'algorithme doit s'initialiser).

La fonction *optim* se déclare de la sorte :

```
[f,uopt]=optim(costf,u0)  ;
```

Ici cette déclaration ne comporte que les deux arguments nécessaires

– `u0` est le vecteur d'initialisation ;
– `costf` est une macro SCILAB, renvoyant la valeur du critère, ainsi que, si possible, celle du gradient du critère par rapport à la commande.

Si on ne précise pas une expression du gradient du critère par rapport à la commande en tout point du domaine d'admissibilité, *optim* procède à une évaluation numérique de celui-là ; cependant pour des raisons de rapidité, et surtout pour assurer la convergence de l'algorithme dans les cas complexes, il est préférable de lui en fournir l'expression.

Une nouvelle primitive : *eval_critère*

Il nous faut donc fournir à la fonction *optim* le gradient de notre critère $\mathcal{J}$, et nous allons pour cela faire appel à la méthode basée sur l'état adjoint.

Même si cette méthode de calcul de gradient est relativement facile à mettre en place, elle est très itérative ; en effet, il faut tout d'abord procéder à l'évaluation de l'état, puis à l'évaluation de l'état adjoint, puis enfin, au calcul du gradient au point voulu. De plus, cette opération va devoir être effectuée des centaines de fois au cours d'une optimisation, à chaque fois que la fonction *optim* voudra évaluer $\nabla\mathcal{J}(u)$.

Or, SCILAB est un langage interprété, qui n'est pas optimisé pour ce genre de tâche très répétitive ; il est donc nécessaire de développer une routine de calcul de gradient *via* le calcul de l'état adjoint en langage C, et pour cela nous allons créer une primitive.

Ainsi la fonction *eval_critère* est une primitive écrite en C, évaluant le critère $\mathcal{J}$ ainsi que son gradient ; sous SCILAB, elle se présente ainsi :

```
[J,grd,etat]=eval_critere(u,x0)
```

On fournit à *eval_critère* le vecteur $u \in \mathbb{R}^{p(T-1)}$ comportant l'ensemble des commandes u_t de dimension p, pour $t \in \{0, \ldots, T-1\}$, ainsi que $x_0 \in \mathbb{R}^n$ l'état initial du système, et *eval_critère* renvoie la valeur `J` du critère, `grd` $\in \mathbb{R}^{p(T-1)}$ le gradient du critère au point u.

Cependant, même si cela n'apparaît pas de manière explicite dans sa déclaration, la primitive *eval_critère* dépend aussi des fonctions ¡¡données ¿¿ du problème, à savoir les fonctions coûts et dynamique. Nous verrons plus loin comment transmettre ces fonctions à *eval_critère*.

Toute ces opérations sont regroupées dans une seule macro SCILAB, nommée *dynoptim*, dont la structure SCILAB est

```
[Jopt,uopt,xopt]=dynoptim(u0,ub,uh,x0)
```

où,

- `u0`, `ub` et `uh` sont des vecteurs de $\mathbb{R}^{p(T-1)}$, correspondant respectivement à la valeur initiale de la commande u utilisée par l'algorithme de minimisation, à la borne inférieure du domaine d'admissibilité pour la commande u, et à la borne supérieure de ce même domaine ;
- `x0` est un vecteur de $\mathbb{R}^n$ correspondant à l'état initial du système ;
- `Jopt` est la valeur minimale du critère ;
- `uopt` $\in \mathbb{R}^{p(T-1)}$ est la commande optimale (*i.e.* permettant de minimiser le critère) ;
- `xopt` $\in \mathbb{R}^{nT}$ est l'état du système correspondant à la commande optimale `uopt` ainsi qu'à l'état initial `x0`.

Étape 1	*optim* initialise le vecteur $u \in \mathbb{R}^{p(t-1)}$ courant avec la valeur `u0` fournie.
Étape 2	*eval_critère* calcule la valeur du critère, ainsi que le gradient du critère au point courant u.
Étape 3	*optim* récupère la valeur du critère ainsi que celle du gradient, puis décide du pas à suivre au cœur de l'algorithme utilisée. Soit l'algorithme n'a pas encore convergé, et on retourne à l'étape 2, soit l'algorithme a convergé, et on passe à l'étape 4.
Étape 4	Une fois que l'algorithme d'optimisation a convergé, *optim* renvoie la commande optimale `uopt`.

TAB. 2.1 – Fonctionnement de la fonction SCILAB *optim*

Étape 1	La primitive *eval_critère* récupère les arguments u et `x0` que lui transmet SCILAB.	*doc_int.c*
Étape 2	*eval_critère* calcule l'état $x = \left(\ x_0, \ \ \ldots, \ \ x_T \ \right) \in \mathbb{R}^{nT}$ correspondant à l'état initial `x0` et à la commande u, (en utilisant la dynamique F).	*dynoptim.c*
Étape 3	*eval_critère* calcule la valeur du critère $\mathcal{J}(u) = J(u, x)$ correspondant à l'état x et à la commande u précédemment calculés.	*dynoptim.c*
Étape 4	*eval_critère* calcule l'état adjoint $\Lambda = \left(\ \Lambda_0, \ \ \ldots, \ \ \Lambda_T \ \right) \in \mathbb{R}^{nT}$ correspondant à l'état x et à la commande u.	*dynoptim.c*
Étape 5	*eval_critère* calcule le gradient du critère au point u en utilisant l'état x et l'état adjoint Λ.	*dynoptim.c*
Étape 6	*eval_critère* renvoie la valeur du critère $\mathcal{J}(u)$, ainsi que celle du gradient au point u, $\nabla\mathcal{J}(u)$.	*doc_int.c*

TAB. 2.2 – Fonctionnement de la primitive eval_critère permettant de calculer la valeur du critère $\mathcal{J}$, ainsi que celle de son gradient $\nabla\mathcal{J}$.

2.3 La création d'une interface permettant l'utilisation de macros SCILAB par l'algorithme de calcul de l'état adjoint

Le choix entre le codage en dur des fonctions critères et dynamique, et l'interfacage avec SCILAB

Nous avons vu que pour des raisons de temps d'exécution, l'algorithme de calcul du gradient du critère $\mathcal{J}$ a été programmé en langage C, au sein de la primitive *eval_critère*.

Cette primitive sert à calculer la valeur du critère, ainsi que son gradient en un point donné, son fonctionnement étant rappelé dans le tableau (2.2).

Pour mener ces calculs, la primitive *eval_critère* doit connaître les fonctions nécessaires à la définition du problème de *Bolza*, à savoir

- la fonction dynamique $F \in \mathcal{F}(\mathbb{R}^p \times \mathbb{R}^n \times \mathbb{N}, \mathbb{R}^n)$ ainsi que ses dérivées par rapport à la commande u, et à l'état x ;
- $l \in \mathcal{F}(\mathbb{R}^p \times \mathbb{R}^n \times \mathbb{N}, \mathbb{R}^n)$, la fonction coût sous le signe somme (2.2), ainsi que ses dérivées par rapport à la commande u, et à l'état x ;
- la fonction coût final $\Phi \in \mathcal{F}(\mathbb{R}^n \times \mathbb{N}, \mathbb{R}^n)$ ainsi que ses dérivées par rapport à la commande u, et à l'état x.

Pour pouvoir fournir ces fonctions à la primitive *eval_critère* qui, rappelons-le est programmée avec le langage C, il y a deux solutions envisageables :

- le codage des fonctions ¡¡en dur¿¿ *i.e.* en langage C ;
- le codage des fonctions sous forme de macros SCILAB, qui seront ensuite transmises à la fonction *eval_critère via* une nouvelle interface entre SCILAB et C.

La première solution offre de gros avantages pour l'écriture des programmes car elle ne nécessite pas d'interface ; c'est aussi cette absence d'interface entre SCILAB et le langage C qui permet d'énormes gains de temps à l'exécution. En effet nous avons vu qu'au cours d'une optimisation toutes les fonctions définissant le problème de *Bolza* (le critère, la dynamique ...) pouvaient être appelées des milliers de fois ; nous avons donc intérêt à limiter au maximum le temps d'accès à ces fonctions ; le codage ¡¡en dur¿¿ est la meilleure réponse à cette exigence.

En revanche, la seconde solution, qui consiste à interfacer SCILAB et le code C, est plus gourmande en temps. Cependant elle est beaucoup plus souple d'utilisation : en effet lorsqu'il veut changer une donnée du problème, l'utilisateur n'a pas à modifier le code C de la fonction *eval_critère*, il doit simplement redéfinir une macro SCILAB, ce qui est quand même beaucoup plus simple.

Les deux modes d'accès aux fonctions ont été implémentés.

En ce qui concerne le premier mode d'accès, à savoir le codage en dur des fonctions, la structure interne des programmes utilisés est celle qui est décrite dans les tableaux (2.2) et (2.1), les fonctions étant définies comme de simples fonctions C, dans le fichier dénommé *fonctions.c*. Ainsi la fonction coût est définie de la manière suivante :

```
void L(double* cmd,double* etat,int* temps,double* res,
                int* n_cmd,int* n_etat).
```

Comme nous pouvons le constater sur cet exemple la déclaration de la fonction critère l n'est pas une mince affaire car il faut non seulement déclarer tous les paramètres et variables de la fonction, mais il faut aussi préciser la dimension de tous ces éléments ; c'est le rôle des variables `n_cmd`, et `n_para`.

Une primitive permettant de fournir à *eval_critère* le nom des macros SCILAB à utiliser : *setf*

En revanche, pour permettre l'utilisation de macros SCILAB par la primitive *eval_critère*, il a fallu modifier de manière conséquente la structure des programmes. En effet les fonctions nécessaires à la définition du problème de *Bolza* ne sont plus écrites dans un fichier C comme auparavant, mais dans un fichier que seul SCILAB peut comprendre ; pour pouvoir les appeler depuis des programmes écrits en C, il faut utiliser la fonction interface *scistring*.

Cette fonction *scistring* permet d'utiliser des macros SCILAB chargées en mémoire, depuis des primitives écrites en C. Il suffit de lui fournir le nom de la macro que l'on veut utiliser, ainsi que les arguments que SCILAB devra transmettre à cette macro, et on récupère les valeurs renvoyées après exécution de la macro.

Il se pose alors le problème de la transmission à la primitive *eval_critère* des noms des fonctions qu'elle devra utiliser lors des calculs : en effet, pour une plus grande souplesse d'utilisation, nous ne pouvons pas fixer définitivement les noms des fonctions à utiliser : la macro SCILAB représentant le coût sous le signe somme (2.2) pourrait aussi bien se nommer `l` que `j`.

Il faut donc stipuler à la primitive *eval_critère* le nom des macros SCILAB qu'elle est censée utiliser, et c'est à cela que sert la primitive *setf*.

Elle s'utilise de la manière suivante

```
setf(nom_L,nom_L_cmd,nom_L_etat,nom_f,nom_f_cmd,
        nom_f_etat,nom_phi,nom_phi_etat),
```

et sert à écrire dans des variables connues de la primitive *eval_critère*, les nom des fonctions à utiliser.

Nous retrouvons donc huit arguments dans la déclaration de la primitive *setf*, à savoir les noms des huit fonctions nécessaires à la définition du problème de *Bolza* :

- `nom_L`, `nom_L_cmd`, `nom_L_etat` sont respectivement les noms des macros SCILAB représentant l, $\dfrac{\partial l}{\partial u}$ et $\dfrac{\partial l}{\partial x}$ où l désigne le coût sous le signe somme (2.2) ;
- `nom_phi`, `nom_phi_cmd`, `nom_phi_etat` sont respectivement les noms des macros SCILAB représentant Φ, $\dfrac{\partial \Phi}{\partial u}$ et $\dfrac{\partial \Phi}{\partial x}$ où Φ désigne le coût final (2.3) ;

– et enfin, `nom_f`, `nom_f_cmd`, `nom_f_etat` sont respectivement les noms des macros SCILAB représentant F, $\frac{\partial F}{\partial u}$ et $\frac{\partial F}{\partial x}$ où F désigne la dynamique (2.4).

Pour une plus grande facilité d'utilisation, la macro SCILAB *dynoptim* regroupe toutes les opérations nécessaires à la bonne marche de la procédure d'optimisation, et sa structure SCILAB est maintenant :

```
[uopt,eopt]=dynoptim(nom_L,nom_L_cmd,nom_L_etat,nom_f,
nom_f_cmd,nom_f_etat,nom_phi,nom_phi_etat,ub,uh,u0,x0)
```

– les huit premiers paramètres étant définis ci-dessus ;
– `u0`, `ub`, et `uh` étant des vecteurs de $\mathbb{R}^{p(T-1)}$, correspondant respectivement à la valeur initiale de la commande u utilisée par l'algorithme de minimisation, à la borne inférieure du domaine d'admissibilité pour la commande u, et à la borne supérieure de ce même domaine ;
– `x0` un vecteur de $\mathbb{R}^n$ correspondant à l'état initial du système dynamique ;
– `Jopt` est la valeur minimale du critère ;
– `uopt` $\in \mathbb{R}^{p(T-1)}$ désignant la commande optimale(*i.e.* permettant de minimiser le critère) ;
– et `xopt`$\in \mathbb{R}^{nT}$ l'état correspondant à la commande `uopt` et à l'état initial `x0` ;
– `eopt` est l'état du système dynamique à l'optimum.

Le fonctionnement de la macro *dynoptim* est résumé dans le tableau suivant :

Étape 1	L'utilisateur charge sous SCILAB les fonctions nécessaires à la définition du problème de *Bolza* , au moyen de l'instruction *getf*.
Étape 2	*setf* fournit à la primitive *eval_critère* les noms des macros SCILAB qu'elle devra utiliser.
Étape 3	Fonctionnement de l'algorithme d'optimisation décrit dans le tableau 2.1.
Étape 4	Une fois que l'algorithme d'optimisation a convergé, la macro *dynoptim* renvoie la valeur de la commande optimale `uopt`, le critère optimal `Jopt`, et `eopt` l'état du système, correspondant à la commande `uopt` et à l'état initial `x0`.

TAB. 2.3 – Fonctionnement de la macro SCILAB *dynoptim* en tenant compte de la primitive d'interface *setf*.

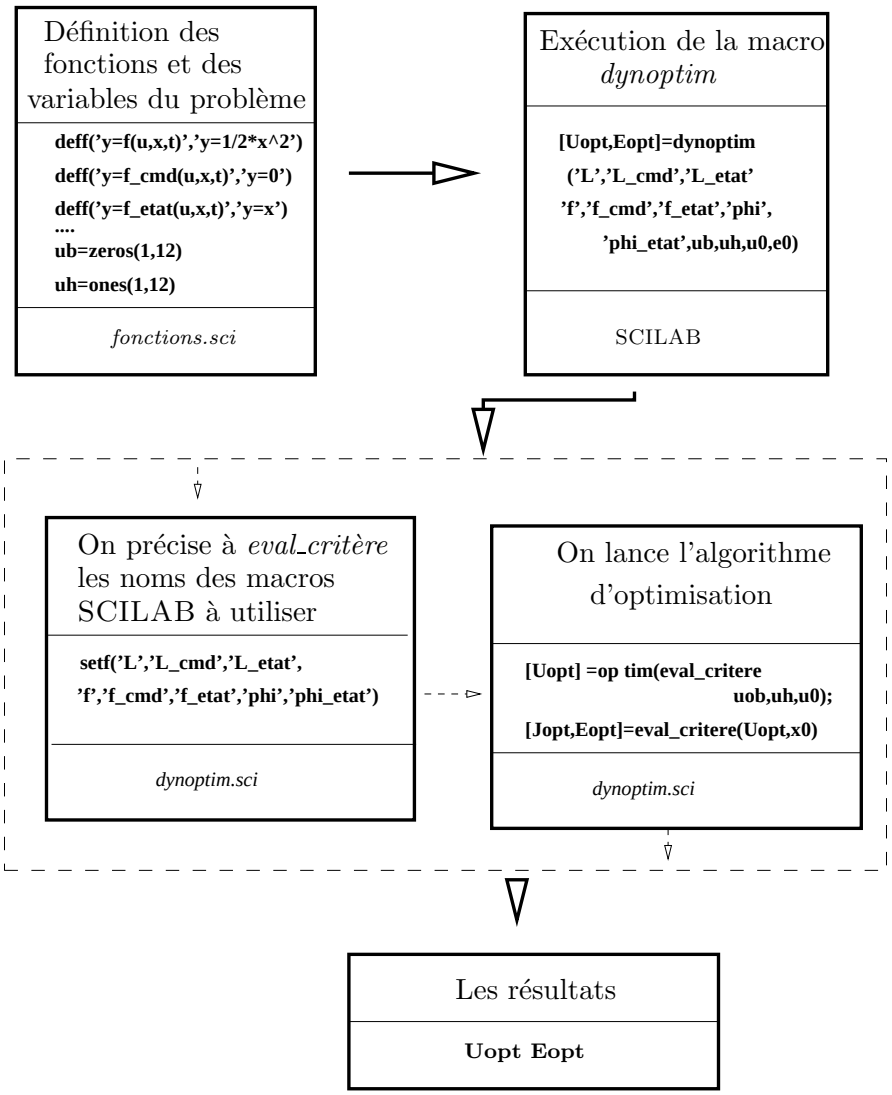

FIG. 2.1 – Fonctionnement de la macro SCILAB *dynoptim*

Le code de la macro SCILAB *dynoptim* est présenté ci-dessous.

```
################################  DYNOPTIM  ###################################


//-------------------------------------------------------------------------
//                        optimisation dynamique
//  version sans contrainte
//          avec primitives+
//-------------------------------------------------------------------------

//nom_... sont les noms des fonctions utilisees par le probleme qui consiste
// a trouver min J(u) avec J(u)=somme( L(u[t],x[t],t))+phi(u[T],x[T],T)
//           0<u<1               t=0..T-1
// en tenant compte de la dynamique :  x[t+1]=f(u[t],x[t],t)


//-------------------------------------------------------------------------


function
[uopt,eopt]=dynoptim(nom_L,nom_L_cmd,nom_L_etat,nom_f,nom_f_cmd,
nom_f_etat,nom_phi,nom_phi_etat,xb,xh,x0,e0)

// on fixe les fonctions a utiliser
//ATTENTION l'odre est le suivant :
setf(nom_L,nom_L_cmd,nom_L_etat,nom_f,nom_f_cmd,
nom_f_etat,nom_phi,nom_phi_etat);


[fopt,uopt,gopt]=optim(dyn_ora,'b',xb,xh,x0,'ar',200,200);

[fopt,gopt,eopt]=eval_critere(uopt,e0);


//-------------------------------------------------------------------------

// une fonction qui fait l'intermediaire entre J et optim
//  raison de compatibilite d' arguments et de retour


function [fo,go,ind]=dyn_ora(x,ind)
[fo,go,eo]=eval_critere(x,e0);
```

2.4 Un petit exemple d'utilisation de la primitive *dynoptim*

Nous voulons ici résoudre le problème suivant.
On cherche

$$\min_{\substack{u_0,\dots,u_{11}\in\mathbb{R}\\ x_0,\dots,x_{12}\in\mathbb{R}^2}} J(u,x),$$

$$\text{avec} \quad J(u,x)=\tfrac{1}{2}\sum_{t=0}^{11} x^2,$$

en tenant compte de la dynamique

$$x_t = u_{t-1}.$$

Il suffit alors de taper les instructions suivantes dans un fichier et de les exécuter sous SCILAB.

```
//*****************************************************
//*                                                   *
//*                Un petit exemple                   *
//*                                                   *
//*****************************************************



//-------------------------- L et ses gradients-------------------------------

deff('y=L(u,x,t)','y=(0.5*x^2)')

deff('y=L_cmd(u,x,t)','y=0')

deff('y=L_etat(u,x,t)','y=x')


// ------------------------f et ses gradients---------------------------------


deff('y=f(u,x,t)','y=u')

deff('y=f_etat(u,x,t)','y=0')

deff('y=f_cmd(u,x,t)','y=1')
```

```
// ------------------------phi et ses gradients--------------------------------

deff('y=phi(x,t)','y=0')

deff('y=phi_etat(x,t)','y=0')


// definition des parametres du probleme
e0=[1];
xb=zeros(1,12);
xh=zeros(1,12)+1;
x0=rand(1,12);


[Uopt,Eopt]=dynoptim("L","L_cmd","L_etat","f","f_cmd","f_etat","phi","phi_etat"
,xb,xh,x0,e0);
```

Après exécution, SCILAB nous donne comme résultat :

```
-->Uopt
 Uopt  =


         column 1 to 6

!   0.    3.331E-16    0.    0.    3.331E-16    3.331E-16 !

         column  7 to 12

!   3.331E-16    3.331E-16    4.441E-16    0.    0.    0.6623569 !

-->Eopt
 Eopt  =


         column  1 to 10

!   1.    0.    0.    0.    0.    0.    0.    0.    0.    4.441E-16 !

         column 11 to 13
```

```
!   0.    0.    0.6623569 !
```

Ce résultat est bien le résultat que nous pouvions espérer, car le problème revenait à minimiser en $u_0, \ldots, u_{11} \in [0,1]$ la fonction :

$$J(u) = \frac{1}{2}\left(x_0^2 + \sum_{t=0}^{11} u_t^2\right)$$

et le minimum de cette fonction est bien atteint pour une commande u vérifiant

$$\begin{cases} \forall t \in \{0, \ldots, 11\} \quad u_t = 0 \\ u_{12} \in [0,1] \end{cases}$$

Ici la commande optimale donnée par SCILAB vérifie bien cette contrainte, et u_{12} est en fait égal à $u0_{12}$ qui est, rappelons le, le 12^{e} de la commande initiale fournie à l'algorithme d'optimisation.

Chapitre 3

Développement d'une primitive SCILAB pour la résolution du problème de Bolza avec contraintes d'état

3.1 Premier essai : une méthode de pénalisation ; problème d'instabilité de l'algorithme d'optimisation

Cette première partie du chapitre va nous familiariser avec les problèmes d'optimisation de *Bolza* sous contrainte d'état.

Nous allons étudier le cas d'un problème de *Bolza* avec les mêmes hypothèses de simplification qu'au paragraphe (2.1.1), mais avec une contrainte de borne sur l'état, *i.e.* un problème de la forme :

Soit $T > 0$ (éventuellement $T = +\infty$), on minimise en $x_0, \ldots, x_T \in \mathbb{R}^n$, et en $u_0, \ldots, u_{T-1} \in \mathbb{R}^p$, le critère

$$J(x,u) = J_1(x,u) + J_2(x,u) \text{ avec,} \tag{3.1}$$

$$\text{le coût intégral} \qquad J_1(x,u) = \sum_0^{T-1} l(x_t, u_t, t), \tag{3.2}$$

$$\text{le coût final} \qquad J_2(x,u) = \Phi(x_T, T)\ , \tag{3.3}$$

en tenant compte de la dynamique

$$x(t+1) = F(x_t, u_t, t), \quad t \in \{0, \ldots, T-1\}, \tag{3.4}$$

éventuellement d'un domaine d'admissibilité

$$u_t \in \mathcal{U} \text{ pour tout } t \in \{0, \ldots, T-1\}, \tag{3.5}$$

et surtout de la contrainte de borne sur l'état

$$\underline{x} \leq x \leq \overline{x}. \tag{3.6}$$

Le problème qui se pose à nous maintenant est de savoir comment traiter cette contrainte de borne sur l'état. Étant donné que, sans cette contrainte de borne sur l'état, nous savons parfaitement résoudre le problème posé, nous avons intérêt à nous ramener à un problème d'optimisation sans contrainte d'état.

Ainsi, pour traiter un problème d'optimisation dynamique, nous nous sommes ramené à un problème d'optimisation statique, et maintenant pour traiter un problème d'optimisation avec contrainte, nous allons essayer de nous ramener à un problème d'optimisation sans contrainte.

3.1.1 Une idée simple : la pénalisation de la contrainte

La pénalisation de la contrainte est effectivement un moyen simple de ramener un problème d'optimisation sous contrainte à un problème d'optimisation sans contrainte.

L'idée est la suivante. Supposons que nous devons résoudre le problème consistant à trouver

$$\min_{u \in \mathcal{U}} J(u), \tag{3.7}$$

en tenant compte de la contrainte

$$u \leq 0. \tag{3.8}$$

Et bien nous allons résoudre le problème qui consiste à trouver u permettant d'atteindre

$$\min_{u \in \mathcal{U}} \left(J(u) + pu \right), \tag{3.9}$$

où p est un réel choisi arbitrairement ¡¡grand¿¿.

En effet si p est suffisamment ¡¡grand¿¿, la seule solution pour l'algorithme d'optimisation– s'il veut minimiser correctement $(J(u) + pu)$– consiste à se placer dans les u négatifs.

La contrainte sera automatiquement vérifiée : on aura bien $u \leq 0$.

Ainsi, nous avons trouvé un moyen de se ramener à un problème d'optimisation sans contrainte, car la contrainte (3.8) a bien disparu dans le problème d'optimisation (3.9). Cependant, cette méthode, bien qu'établie sous des hypothèses précises (Culioli, 1994) , présente des problèmes de mise en place, comme en témoigne l'exemple suivant.

Nous allons essayer de résoudre par une méthode de pénalisation le problème suivant :

$$\begin{array}{l} \min \\ x \in [-5,5] \\ y \in [-5,5] \end{array} \quad x^2 - 2y^2, \tag{3.10}$$

sous la contrainte

$$x + y = 1. \tag{3.11}$$

Ce problème est très simple et le couple solution se calcule à la main : on trouve

$$\begin{aligned} x_{\text{opt}} &= 2 \\ y_{\text{opt}} &= -1 \end{aligned}$$

Essayons maintenant de résoudre ce problème avec SCILAB, en utilisant la fonction *optim*, et une pénalisation de la contrainte : le critère à minimiser sera

$$j(x,y) = x^2 - 2y^2 + p(1 - x - y)^2. \tag{3.12}$$

On définit dans le code qui suit la fonction *costf* qui renvoie le critère (dans lequel nous avons déjà inclu la pénalité p de la contrainte), ainsi que son gradient, puis on demande à SCILAB de minimiser ce critère en partant du point $x0$.

```
//------------------ definition du critère et de son gradient --------------


deff('[f,g,ind]=costf(x,ind)','f=2*x(1)^2-x(2)^2+p*(1-x(1)-x(2))^2,g=zeros(1,2)
,g(1)=4*x(1)-2*p*(1-x(1)-x(2)),g(2)=-2*x(2)-2*p*(1-x(1)-x(2))');

// le point initial
x0=10*rand(1,1);

//------------------- optimisation -----------------------------------------
```

```
[fopt,xopt]=optim(costf,x0)
```

Le problème de cette méthode vient du fait qu'il faut parfaitement ajuster la valeur de la pénalité p, les résultats obtenus sont regroupés dans le tableau 3.1 :

$p = 1$	`xopt = 1.0E+95 * ! - 10. 30. !`
$p = 10$	`xopt = ! - 1.25 2.5 !`
$p = 10$	`xopt = ! - 1.0204082 2.0408163 !`
$p = 1000$	`xopt = ! - 1.002004 2.004008 !`

TAB. 3.1 – Résultats d'un exemple de pénalisation de la contrainte

Il apparaît clairement que, pour que cette méthode fonctionne, il faut que la pénalité p ne soit pas trop petite. Cependant, nous n'avons pas non plus intérêt à prendre une pénalité p trop grande, car, dans le cas de problèmes compliqués, les gradients des fonctions en jeu ont des normes trop importantes, et donc les algorithmes d'optimisation ne fonctionnent plus correctement.

Ainsi pour qu'une méthode de pénalisation de la contrainte soit efficace, il faut que la pénalité soit ajustée à la main en fonction du problème étudié, ce qui peut être parfois très ennuyeux : on ne peut pas créer, comme nous voulons le faire, un outil transposable à de nombreux problèmes.

Il existe des méthodes beaucoup plus souples, pouvant être transposées à des problèmes différents sans nécessiter pour autant de phases de ¡¡règlage¿¿ trop importantes. Ce sont des méthodes où on introduit des critères plus complexes basés sur des multiplicateurs associés aux contraintes : on parle alors de *Lagrangien*.

3.2 Les méthode de Lagrangien

Les méthodes de *Lagrangien*, ou méthodes duales reposent sur la maximisation de la fonction duale $H(p)$ dans $\mathbb{R}^m$, où H est définie de la façon suivante :

On introduit le *Lagrangien*

$$L(u, p) = \mathcal{J}(u) + p'\theta(u),$$

où θ désigne la fonction de contrainte : par exemple pour une contrainte de type

$$u \leq \overline{u},$$

θ est la fonction

$$\theta(u) = u - \overline{u}.$$

On définit alors

$$H(p) = \min_u L(u, p),$$

où la minimisation est faite à p fixé.

La maximisation de $H(p)$ est basée sur le fait que, moyennant des hypothèses de régularité et de convexité convenables pour toutes les fonctions en jeu (Culioli, 1994), on dispose de l'identité

$$\nabla H(p) = \theta(u(p)), \tag{3.13}$$

avec

$$u(p) = \arg\min_u \left(\mathcal{J}(u) + p'\theta(u)\right).$$

En appliquant l'algorithme *d'Uzawa* ou celui *d'Arrow-Hurwicz* (Culioli, 1994), on trouve le couple (u^*, p^*) qui est solution de notre problème.

Cependant, cette méthode nécessite des hypothèses de convexité assez forte pour la fonction contrainte afin d'obtenir l'identité de point selle (Cohen, 1999)

$$\max_p \min_u L(u, p) = \min_u \max_p L(u, p). \tag{3.14}$$

On fait alors appel à une forme de Lagrangien plus complexe, appelé *Lagrangien augmenté* (Culioli, 1994), permettant de s'affranchir de certaines hypothèses de convexité des fonctions entrant en jeu dans le problème, ceci afin de développer un outil le plus général possible.

3.3 Une méthode de *Lagrangien augmenté* appliquée à la structure du problème dynamique de *Bolza*

Nous allons préciser ici la méthode de résolution utilisée dans *dynoptimsc*, un programme SCILAB de résolution de problème dynamique sous contrainte de borne d'état.

3.3.1 Le Lagrangien augmenté avec contrainte d'inégalité

Ce paragraphe est seulement une courte présentation des outils que nous allons utiliser pour pouvoir résoudre le problème qui nous est posé. Toutes les démonstrations ainsi que les justifications de ce qui est avancé ici figurent dans le cours de M. Guy Cohen *Optimisation des grands systèmes, cours de* DEA *1999*.

Lorsque on veut résoudre un problème d'optimisation sous contrainte d'inégalité

$$\min_u \mathcal{J}(u), \tag{3.15}$$

sous la contrainte

$$\underline{u} < u < \overline{u}, \tag{3.16}$$

le Lagrangien augmenté associé est le suivant :

On désigne par θ, le vecteur contrainte, *i.e.* si u est un vecteur de dimension p,

$$\theta(u) = \begin{pmatrix} u_0 - \overline{u_0} \\ \underline{u_0} - u_0 \\ \vdots \\ u_{p-1} - \overline{u_{p-1}} \\ \underline{u_{p-1}} - u_{p-1} \end{pmatrix}. \tag{3.17}$$

On désigne par m, dans ce qui suit, la taille du vecteur contrainte ; ici dans le cas de contrainte d'inégalité de type (3.16), $m = 2p$. Il est à noter que $\theta(u)$ et la variable duale p sont de même dimension m.

Le lagrangien augmenté s'écrit alors :

$$L_c(u,p) = \mathcal{J}(u) + \frac{c}{2}\|\text{proj}_{(\mathbb{R}^+)^m}\left(\theta(u) + \frac{p}{c}\right)\|^2 - \frac{1}{2c}\|p\|^2. \tag{3.18}$$

L'algorithme que nous allons utiliser est résumé dans le tableau 3.2 :

Nous avons introduit ici deux paramètres réels ρ et c. Les conditions de convergence révèlent un compromis dans le choix de ces deux paramètres :

- plus c et ρ sont grands, plus les pas de gradients seront grands et l'algorithme convergera plus vite ;
- cependant, pour c trop grand, la minimisation conduite à l'étape 3 sera difficile à réaliser.

Le cours de M Guy Cohen sur *l'Optimisation des grands systèmes* développe beaucoup plus longuement ces conditions de convergence.

Étape 1	On initialise le couple (u, p) avec les valeurs (u^0, p^0) ; on pose $k = 0$.
Étape 2	On résoud $\min_u \left(L(u, p^k)\right)$, soit u^k une solution.
Étape 3	On calcule : $p^{k+1} = \left(1 - \frac{\rho}{c}\right) p^k + \frac{\rho}{c} \text{proj}_{(\mathbb{R}^+)^m} \left(p^k + c\theta(u^{k+1})\right)$.
Étape 4	Si $\|u^{k+1} - u^k\| + \|p^{k+1} - p^k\|$ est suffisamment petit, on s'arrête, sinon on retourne à l'étape 2.

TAB. 3.2 – Algorithme d'Uzawa, appliqué au Lagrangien augmenté, avec contrainte d'inégalité

3.4 L'algorithme utilisé dans *dynoptimsc* : un Lagrangien augmenté sous une forme dynamique

Pour résoudre le problème de *Bolza* sous contrainte, nous avons créé des outils (une primitive SCILAB appelée *eval_critère*, et la macro associée *dynoptim*) capable de calculer les gradients puis de minimiser des fonctions se présentant sous la forme suivante :

$$\mathcal{J}(u) = \sum_{t=0}^{T-1} l(u_t, x_t, t) + \phi(x_T, T) \tag{3.19}$$

avec la dynamique :

$$x_t = F(u_{t-1}, x_{t-1}, t-1) \tag{3.20}$$

Ainsi, afin de pouvoir se baser sur les programmes informatiques déjà existants pour mener à bien **l'étape 2** de l'algorithme décrit dans le tableau (3.2), il serait très intéressant de pouvoir se ramener à un lagrangien augmenté de la forme suivante, à p fixé :

$$L_c(p, u) = \sum_{t=0}^{T-1} l_p(u_t, x_t, t) + \phi_p(x_T, T) \tag{3.21}$$

en tenant compte de la même dynamique (3.20).

Rappel du problème de *Bolza* avec contrainte de bornes sur l'état

Nous voulons résoudre des problèmes de la forme suivante,

Soit $T > 0$ (éventuellement $T = +\infty$), on minimise en $x_0, \ldots, x_T \in \mathbb{R}^n$, et en $u_0, \ldots, u_{T-1} \in \mathbb{R}^p$, le critère

$$J(x,u) = J_1(x,u) + J_2(x,u) \text{ avec,} \tag{3.22}$$

$$\text{le coût intégral} \qquad J_1(x,u) = \sum_0^{T-1} l(x_t, u_t, t)dt \ , \tag{3.23}$$

$$\text{le coût final} \qquad J_2(x,u) = \Phi(x_T, T) \ , \tag{3.24}$$

en tenant compte de la dynamique

$$x(t+1) = F(x_t, u_t, t), \quad t \in \{0, \ldots, T-1\}, \tag{3.25}$$

éventuellement d'un domaine d'admissibilité

$$u_t \in \mathcal{U} \text{ pour tout } t \in \{0, \ldots, T-1\}, \tag{3.26}$$

et de la contrainte de borne sur l'état

$$\underline{x} \leq x \leq \overline{x}. \tag{3.27}$$

Définition du Lagrangien augmenté $L_c(u,p)$, ainsi que du vecteur contrainte $\theta(u)$

Nous allons tout d'abord définir le vecteur contrainte correspondant à la contrainte (3.27). Nous allons procéder comme pour la contrainte (3.17).

Pour la contrainte

$$\underline{x} \leq x \leq \overline{x}, \tag{3.28}$$

on définit le vecteur contrainte θ :

$$\theta(x) = \begin{pmatrix} x_0 - \overline{x_0} \\ \underline{x_0} - x_0 \\ \vdots \\ x_{n-1} - \overline{x_{n-1}} \\ \underline{x_{n-1}} - x_{n-1} \end{pmatrix}. \tag{3.29}$$

où n est, comme convenu, la dimension de l'état du problème.

Le vecteur contrainte $\theta(u)$ est donc un vecteur réel de dimension $m = 2n$.

Nous pouvons maintenant définir le *Lagrangien augmenté* associé à notre problème.

Nous avons déjà vu que ce lagrangien s'écrit

$$L_c(u,p) = \mathcal{J}(u) + \frac{c}{2}\|\text{proj}_{(\mathbb{R}^+)^m}\left(\theta(u) + \frac{p}{c}\right)\|^2 - \frac{1}{2c}\|p\|^2.$$

En développant l'expression de notre critère $\mathcal{J}(u)$, on arrive à l'expression :

$$L_c(u,p) = \sum_0^{T-1} l(x_t,u_t,t) + \Phi(x_T,T) \ + \frac{c}{2}\|\text{proj}_{(\mathbb{R}^+)^m}\left(\theta(u) + \frac{p}{c}\right)\|^2 - \frac{1}{2c}\|p\|^2,$$

puis, en tenant compte de l'expression de notre vecteur contrainte, on obtient :

$$\begin{aligned} L_c(u,p) \ = \ & \sum_0^{T-1}\Bigg(l(x_t,u_t,t) + \frac{c}{2}\Bigg(\max(0, x_t - \overline{x} + \frac{p^{(2t)}}{c})^2 \\ & + \max(0, \underline{x} - x_t + \frac{p^{(2t+1)}}{c})^2\Bigg) - \frac{1}{2c}(p^{(2t)} + p^{(2t+1)})\Bigg) \\ & + \Bigg(\Phi(x_T,T) + \frac{c}{2}\Bigg(\max(0, x_T - \overline{x} + \frac{p^{(2T)}}{c})^2 \\ & + \max(0, \underline{x} - x_T + \frac{p^{(2T+1)}}{c})^2\Bigg) - \frac{1}{2c}(p^{(2T)} + p^{(2T+1)})\Bigg), \end{aligned} \tag{3.30}$$

si bien qu'en définissant les fonctions

$$\begin{aligned} l_p(x_t,u_t,t) \ = \ & l(x_t,u_t,t) \\ + \ & \frac{c}{2}\left(\max(0, x_t - \overline{x} + \frac{p^{(2t)}}{c})^2 + \max(0, \underline{x} - x_t + \frac{p^{(2t+1)}}{c})^2\right) \\ - \ & \frac{1}{2c}\left(p^{(2t)} + p^{(2t+1)}\right) \end{aligned} \tag{3.31}$$

$$\begin{aligned} \phi_p(x_T,T) \ = \ & \phi(x_T,u_T,T) \\ + \ & \frac{c}{2}\left(\max(0, x_T - \overline{x} + \frac{p^{(2T)}}{c})^2 + \max(0, \underline{x} - x_T + \frac{p^{(2T+1)}}{c})^2\right) \\ - \ & \frac{1}{2c}\left(p^{(2T)} + p^{(2T+1)}\right), \end{aligned} \tag{3.32}$$

on se ramène à un lagrangien de la forme voulue, *i.e.*

$$\mathcal{J}(u) = \sum_{t=0}^{T-1} l_p(x_t, u_t, t) + \phi_p(x_T, T). \tag{3.33}$$

Sous cette forme, il sera très facile de procéder à la minimisation de ce Lagrangien à p fixé, lors de l'étape 2 de l'algorithme présenté dans le tableau (3.2), car on dispose déjà de l'outil *dynoptim* pour minimiser les fonctions de la forme (3.33).

Il est à noter que les fonctions l_p et ϕ_p sont toutes deux dérivables, mais à dérivées non continues : l'algorithme *optim* utilisé dans *dynoptim* sera tout de même capable de minimiser $\mathcal{J}$.

Mise en place de l'aspect informatique de l'algorithme

Nous allons programmer une macro SCILAB qui va nous permettre de résoudre des problèmes dynamiques de *Bolza* sous contrainte de borne sur l'état : *dynoptimsc*.

Nous avons vu que pour cela nous allions utiliser l'algorithme présenté dans le tableau (3.2), et que pour se faire nous allions mettre le *Lagrangien augmenté* sous une forme telle qu'il sera possible de mettre à profit l'outil *dynoptim* déjà construit.

Nous rappelons que dans ce qui suit
- une primitive désigne une fonction utilisable depuis SCILAB ;
- une macro désigne une fonction programmée en langage SCILAB, et exécutable depuis SCILAB.

Pour résoudre un problème de *Bolza* sous contrainte de borne sur l'état, nous allons avoir besoin
- de la macro SCILAB *dynoptim* ;
- d'une primitive *theta* qui jouera le rôle de la fonction *vecteur de contrainte* définie au paragraphe 3.2.

Enfin pour des raisons de souplesse d'utilisation, l'ensemble de l'algorithme que nous allons utiliser sera écrit dans la macro SCILAB *dynoptimsc*.

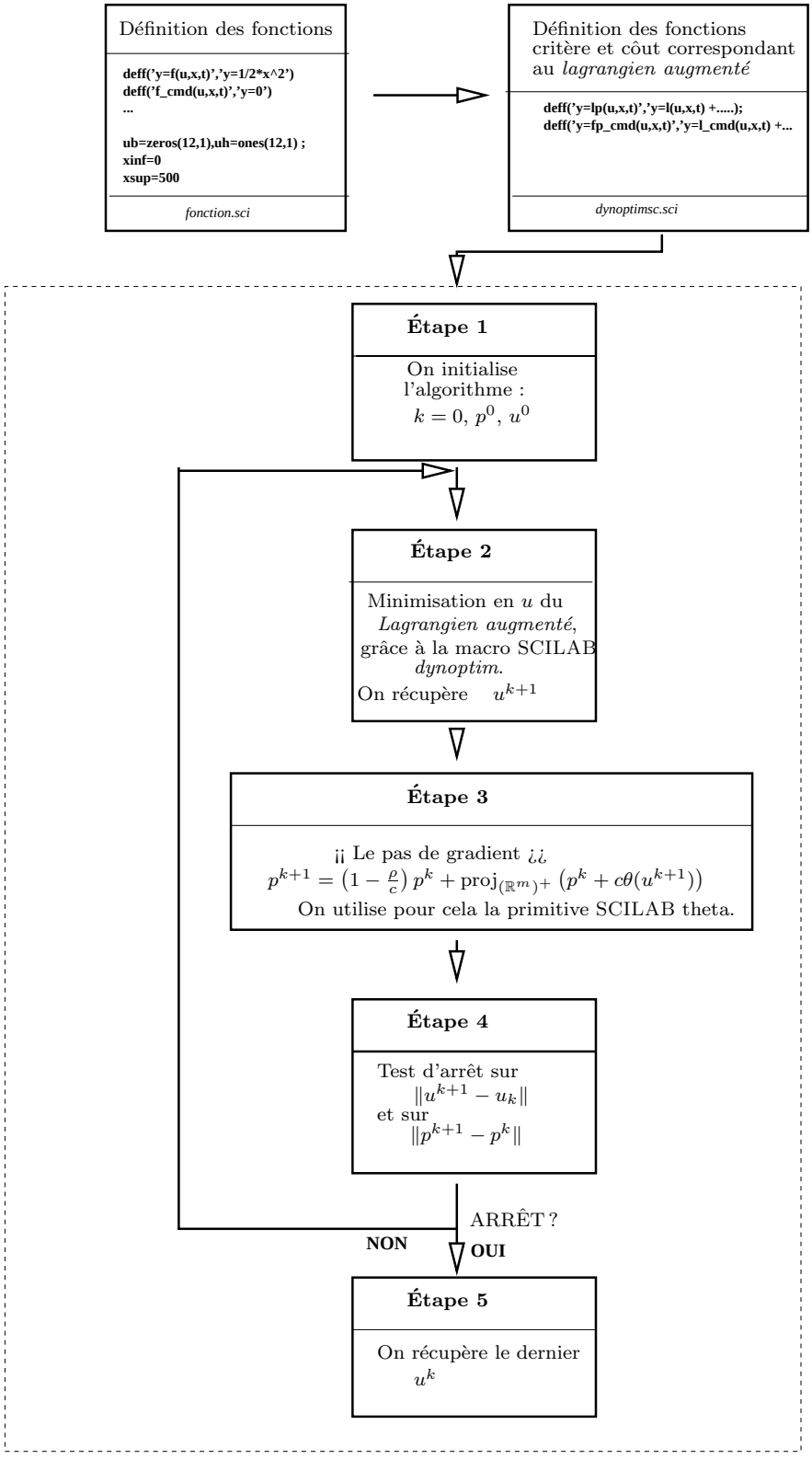

FIG. 3.1 – Fonctionnement de la macro SCILAB *dynoptimsc*

Chapitre 4

Quelques résultats de simulations numériques des modèles STARTS

4.1 Le modèle STARTS d'optimisation sous contrainte environnementale

4.1.1 Un bref rappel du problème posé

Le problème posé consiste à modéliser un pays qui doit mener une politique de réduction des émissions de gaz à effet de serre en deçà d'un seuil préalablement fixé. Cependant, mener une telle politique a un coût, et ce coût doit être minimisé.

Le problème est donc formulé de la manière suivante : l'objectif consiste à minimiser la somme actualisée (à un taux $\rho > 0$) des coûts de réduction des émissions de gaz à effet de serre

$$J(a) = \sum_{t=0}^{T-1} C(a_t, a_{t-1}, t) \frac{1}{(1+\rho)^t}, \tag{4.1}$$

avec un coût d'abattement qui s'écrit

$$C(a_t, a_{t-1}, t) = \alpha \overline{E_t} {a_t}^{\nu} \lambda(t) \gamma(a_t, a_{t-1}), \tag{4.2}$$

sous la contrainte environnementale de non dépassement d'un seuil de concentration en CO_2 préalablement fixé

$$M_t \leq \overline{M}. \tag{4.3}$$

Pour modéliser l' évolution des concentrations en CO_2, nous utilisons la dynamique d'accumulation du CO_2 suivante :

$$\begin{cases} M_{t_{|t=0}} &= M_0 \\ M_t &= M_{t-1} + \Delta t \Big(\beta \overline{E}_{t-1}(1 - a_{t-1}) - \sigma(M_{t-1} - M_{-\infty}) \Big). \end{cases} \tag{4.4}$$

On considère qu'une fraction β des émissions est stockée dans l'atmosphère, et qu'une fraction σ est réabsorbée par le sol et les océans.

On rappelle que le coût d'abattement comporte trois facteurs

- un terme dépendant de a_t en $\alpha\overline{E_t}a_t^\nu$ qui est le coût d'abattement ¡¡brut¿¿ ;
- un terme de progrès technique $\lambda(t)$ qui implique une diminution du coût nominal d'abattement avec le temps ;
- un terme $\gamma(\dot{a}_t)$ qui traduit l'inertie du capital, en pénalisant de trop grands écarts entre a_t, et a_{t-1}.

Dans la plupart des cas, la modélisation fût menée sur douze pas de temps ($T = 12$), et le modèle mathématique utilisé est celui qui est décrit au paragraphe 1.4.

Les valeurs des paramètres utilisés par défaut sont résumées dans le tableau (4.1.1).

E_t	Gigatonnes par an	5.9623, 6.998, 8.4363, 9.9111, 11.018, 12.126, 13.233, 14.541, 15.848, 17.156, 18.463, 19.771
α	dollars par tonne de carbone et par an	1 000
ν	sans dimension	3
M_0	ppm	360
$M_{-\infty}$	ppm	274
ρ	sans dimension	0.05
σ	unité par an	0.01
Δt	an	10
β	ppm par gigatonne par an	0.38

TAB. 4.1 – Définitions et valeurs par défaut des paramètres utilisés dans les modèles STARTS

La fonction de progrès technique $\lambda(t)$ est une fonction décroissante vérifiant $\lambda(0) = 1$; pour les essais numériques ci dessous, elle a été choisie de la forme :

$$\lambda(t) = 0.25 + 0.75 \exp -0.01 \times \Delta t \times t. \tag{4.5}$$

Cette fonction simule les conséquences du progrès technique en faisant décroître le coût des mesures de réductions des émissions avec le temps : on considère que les méthodes utilisées s'améliorent avec le temps, d'où des baisses de coût.

Nous allons procéder à différentes simulations, correspondant chacune à une forme particulière du problème d'*optimisation sous contrainte environnementale* selon

- la forme de la fonction d'inertie γ ;
- la valeur du seuil de concentration $\overline{M}$ à ne pas dépasser.

4.1.2 Une comparaison entre SCILAB et le logiciel GAMS

Le modèle STARTS d'*optimisation sous contrainte environnementale* avait déjà été programmé par M. Franck Lecocq avec le logiciel GAMS. Dans le cadre d'une amélioration de ce modèle, le CIRED a demandé qu'il soit transposé sous SCILAB.

Le fait que nous disposons à la fois du modèle STARTS d'*optimisation sous contrainte environnementale* sous GAMS et sous SCILAB, nous a permis de tester les résultats donnés par *dynoptimsc* sur un modèle concret.

Exceptionnellement, nous nous sommes placés dans la cadre d'un modèle à 11 pas de temps, car c'est ainsi que le programme sous GAMS de Franck Lecocq donne ses résultats.

La fonction d'inertie $\gamma(a_t, a_{t-1})$ est choisie de la forme [1] :

$$\gamma(a_{t-1}, a_t) = \begin{cases} 1 & \text{si } |a_t - a_{t-1}| < \overline{\gamma} \\ \dfrac{|a_t - a_{t-1}|}{\overline{\gamma}} & \text{sinon.} \end{cases}$$

Le seuil environnemental $\overline{M}$ est fixé à 450 ppm et le paramètre $\overline{\gamma}$ prend successivement les valeurs : 0.01 puis 0.005.

Les résultats sont représentés par les courbes ci-dessous montrant l'évolution de l'abattement a_t en fonction du temps. Les résultats donnés par le modèle écrit avec SCILAB sont en bleu, ceux donnés par GAMS sont en rouge.

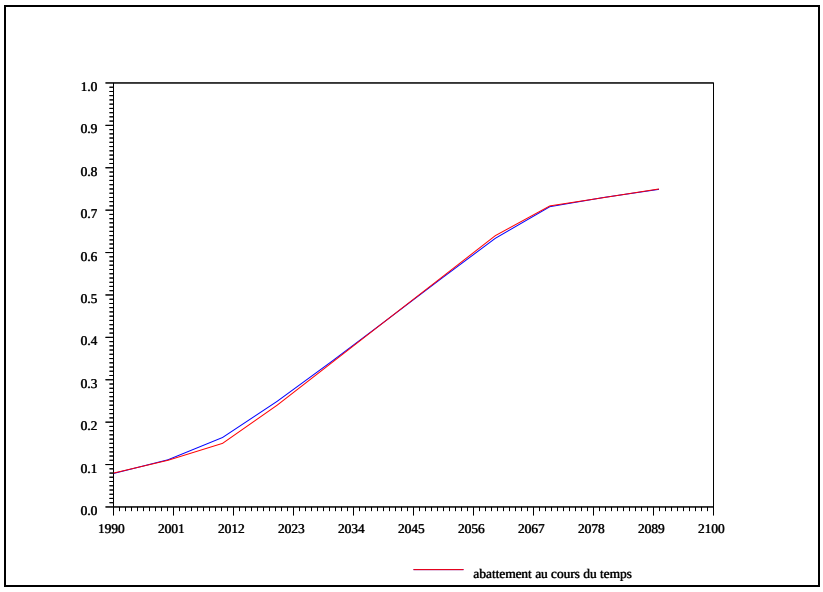

FIG. 4.1 – Trajectoires d'abattement, comparées entre GAMS (rouge) et SCILAB (bleu), pour $\overline{\gamma} = 0.01$

[1] la fonction γ utilisée est lissée pour être dérivable sur tout $\mathbb{R}^2$

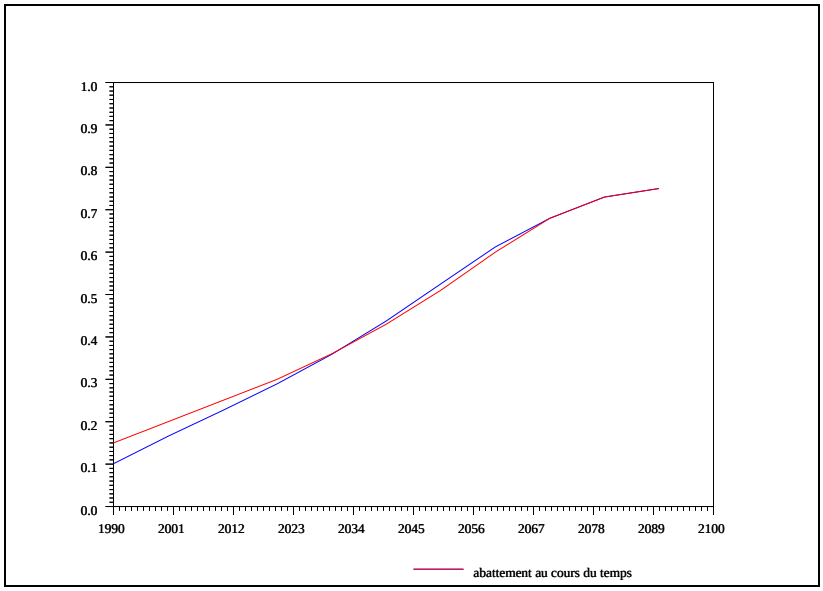

FIG. 4.2 – Trajectoires d'abattement, comparées entre GAMS (rouge) et SCILAB (bleu), pour $\overline{\gamma} = 0.005$

Il apparaît que plus $\overline{\gamma}$ est petit, donc plus on pénalise les variations brutales de la commande entre deux pas de temps, et plus la trajectoire d'abattement est lisse.

Étant donné que grâce au progrès technique λ et grâce à l'actualisation de la monnaie, le coût d'un même abattement diminue au cours du temps, le pays a toujours intérêt à retarder le plus possible la date pour laquelle il commencera à abattre. Ce n'est qu'en introduisant la fonction γ que l'on peut atténuer ce genre de comportement en pénalisant les trop grandes variations de la commande.

On notera que GAMS et SCILAB ne donnent pas exactement les mêmes résultats, avec, cependant, des trajectoires semblables : cela vient en partie du fait que les résultats fournis par GAMS étaient arrondis à deux chiffres après la virgule, ce qui n'est pas sans poser de problème pour des résultats compris entre 0 et 1.

Nous avons donc obtenu des résultats cohérents concernant les conséquences de la variation du paramètre $\overline{\gamma}$, et *dynoptimsc* est apparu au moins aussi performant que le logiciel GAMS pour la résolution de ce problème d'optimisation dynamique sous contrainte de borne d'état : les temps de SCILABd'exécution sont sensiblement meilleurs avec une moyenne de l'ordre de six secondes. Nous reviendrons plus loin sur la comparaison des résultats obtenus.

4.1.3 Influence de la contrainte environnementale $\overline{M}$ sur la trajectoire d'abattement

Nous allons fixer ici $\overline{\gamma} = 0.01$, et nous allons étudier les modifications de la trajectoire d'abattement au cours du temps, en fonction de la valeur du seuil environnemental $\overline{M}$.

Nous allons procéder à plusieurs simulations pour des valeurs de $\overline{M}$ comprises entre 380 ppm et 600 ppm.

Nous nous sommes placés dans le cadre d'un modèle à douze pas de temps, et les différentes trajectoires obtenues sont représentées sur les figures 4.3, 4.4, 4.5, et 4.6.

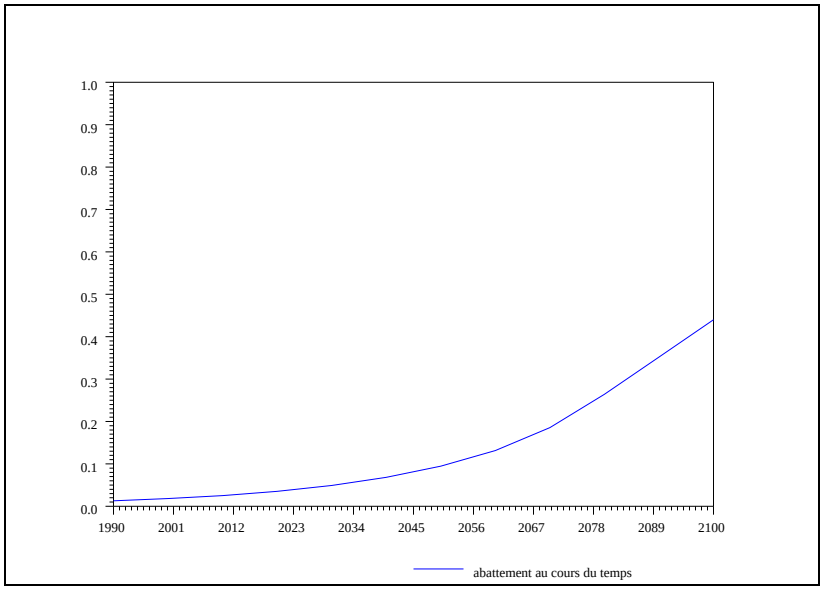

FIG. 4.3 – Trajectoire d'abattement donnée par *dynoptim*, pour $\overline{M} = 600$

Comme nous l'avons déjà remarqué plus haut, le progrès technique et l'actualisation de la monnaie font que le coût de l'abattement diminue avec le temps ; un pays a donc toujours intérêt à retarder le plus possible sa politique d'abattement.

Nous retrouvons bien ce comportement dans les trajectoires d'émissions obtenues ici, d'autant plus qu'avec $\overline{\gamma} = 0.1$ les changements brutaux de décisions d'abattement sont peu pénalisés.

Les courbes présentent trois phases :
- la première phase correspond à la période où le pays a intérêt à attendre avant de réduire les émissions ;
- la seconde phase est la transition : pour ne pas dépasser le seuil environnemental il faut commencer à abattre ;
- enfin la troisième phase correspond à une période de stabilisation de la politique avec un abattement élevé et stable avec le temps.

Selon la valeur de la contrainte environnementale, les durées respectives de ces trois phases varient. Ainsi
- pour $\overline{M} = 600$, la situation n'est pas assez contraignante, et le pays attend longtemps avant d'abattre un peu sur la fin de la période : il n'y a pas de phase de stabilisation finale car l'horizon de calcul est trop proche ;
- pour $\overline{M} = 500$, et pour $\overline{M} = 450$, la contrainte devient active, et les trois phases de la trajectoire d'émissions sont bien visibles ;

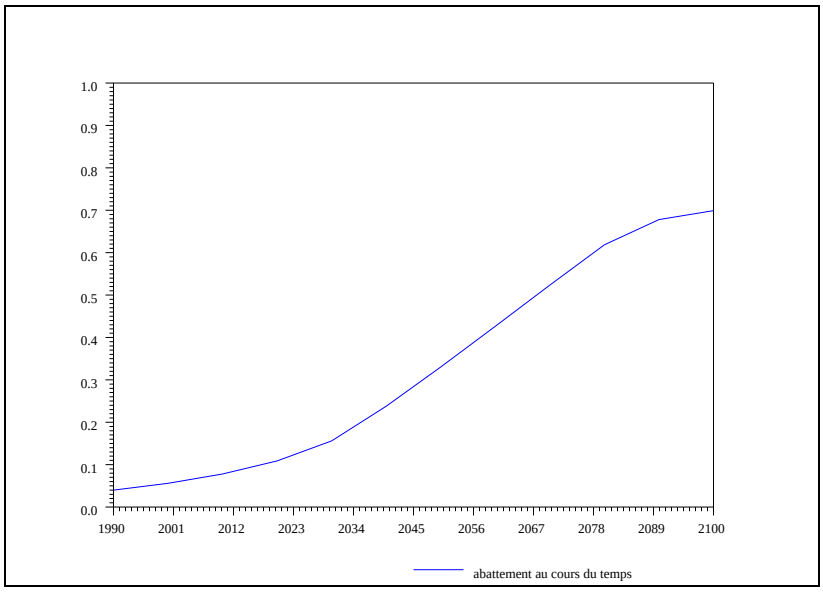

FIG. 4.4 – Trajectoire d'abattement donnée par *dynoptim*, pour $\overline{M} = 500$

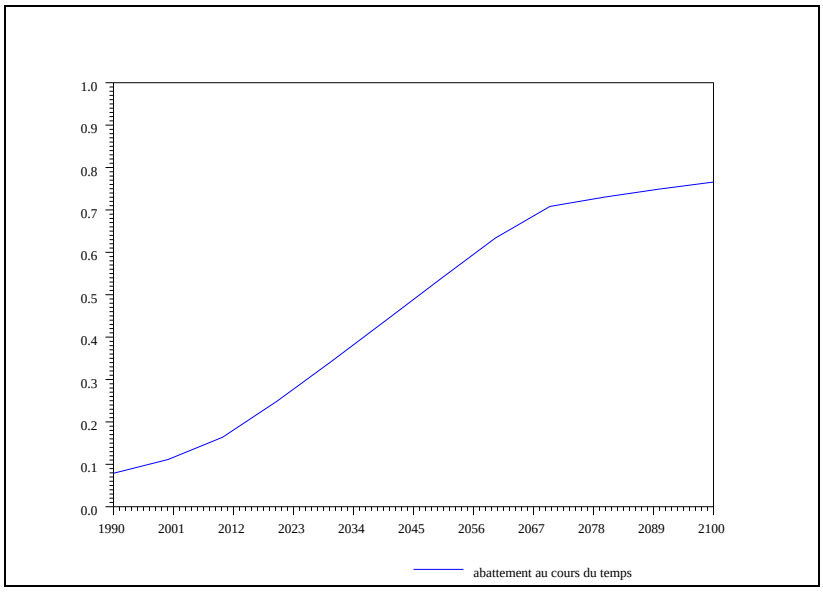

FIG. 4.5 – Trajectoire d'abattement donnée par *dynoptim*, pour $\overline{M} = 450$

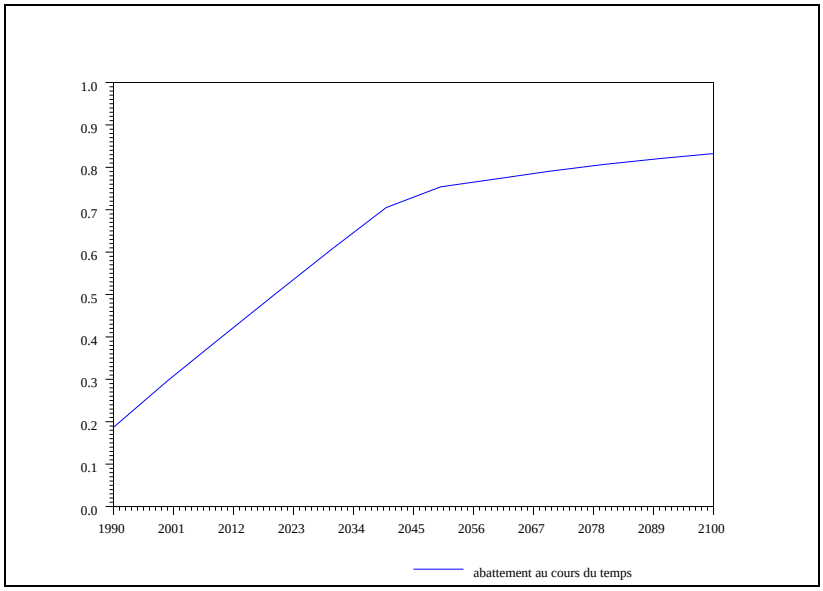

FIG. 4.6 – Trajectoire d'abattement donnée par *dynoptim*, pour $\overline{M} = 400$

– pour $\overline{M} = 400$, nous sommes dans une situation très contraignante, avec un objectif environnemental très difficile à atteindre : il n'y a donc pas de phase ¡¡ d'attente ¿¿, on doit abattre tout de suite pour rester en dessous du seuil de concentration autorisé.

On pourra remarquer sur la figure 4.7 de quelle manière *dynoptimsc* respecte les contraintes : la trajectoire de concentration est tangente à la droite $y = \overline{M}$ quand t se rapproche de T.

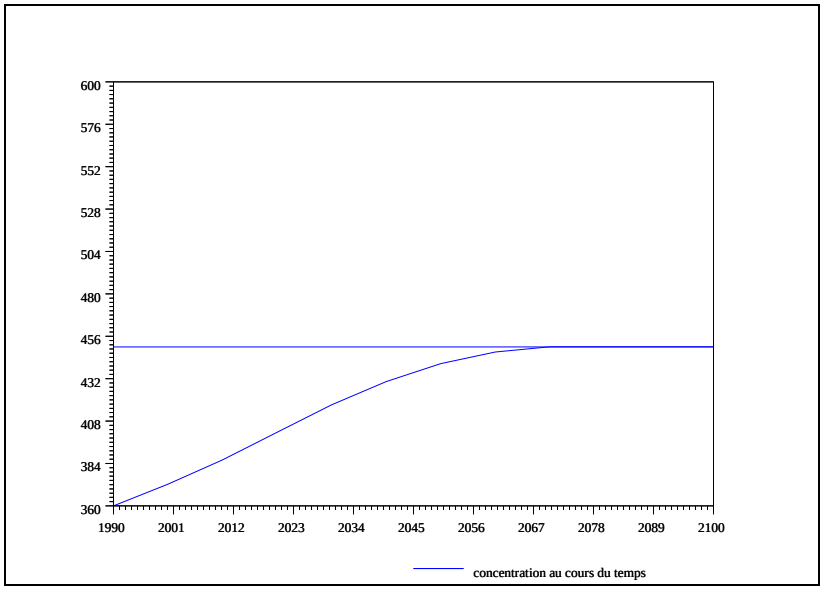

FIG. 4.7 – Trajectoire de concentration, et valeur seuil $\overline{M} = 450$ ppm

4.2 Le modèle STARTS d'optimisation en coût avantage

4.2.1 Rappel du problème posé

Le problème posé consiste maintenant à modéliser un pays qui doit mener une politique de réduction des émissions de gaz à effet de serre, en minimisant à la fois, les coûts

– des impacts futurs des changements climatiques ;

– de la politique de réduction des émissions.

Il existe donc un arbitrage explicite entre les coûts de l'abattement et les coûts des impacts, et cette façon d'aborder le problème est plus rationnelle pour un économiste que le fait d'imposer une contrainte de non dépassement d'une contrainte environnementale.

Concrètement la contrainte environnementale (1.2) a disparu, et maintenant le coût des impacts des dommages dus à la pollution est pris en compte dans la fonction *objectif*.

La dynamique reste inchangée, les états sont liés par la dynamique (4.4), à savoir :

$$\left\{\begin{array}{lcl} M_{t_{|t=0}} & = & M_0 \\ M_t & = & M_{t-1} + \Delta t\Big(\beta\overline{E_{t-1}}(1-a_{t-1}) - \sigma(M_{t-1} - M_{-\infty})\Big), \end{array}\right.$$

et le critère à minimiser est maintenant

$$J(a,M) = \sum_{t=0}^{T-1}\Big(C(a_t,a_{t-1},t) + d(M_t,t)\Big)\frac{1}{(1+\rho)^t}. \tag{4.6}$$

Le terme $C(a_t,a_{t-1},t)$ du coût de l'abattement dans l'expression du critère est identique à l'expression utilisée pour le modèle avec contrainte environnementale ,

$$C(a_t,a_{t-1},t) = \alpha\overline{E_t}{a_t}^{\nu}\lambda(t)\gamma(a_t,a_{t-1}),$$

et le terme correspondant au coût des dommages s'écrit comme le produit du PIB $\kappa(t)$ de référence et d'un indicateur de dommage $\xi(t)$ variant entre 0 et 1, soit :

$$d(M_t,t) = \kappa(t)\xi(M_t). \tag{4.7}$$

Cet indicateur de dommage peut prendre plusieurs formes : il est possible d'envisager des comportements de croissance ¡¡lente¿¿, par exemple linéaire

$$\xi(M_t) = \theta(M_t - M_0), \tag{4.8}$$

où θ est un réel fixé.

Il est aussi possible d'examiner des comportements explosifs, par exemple exponentiel :

$$\xi(M_t) = \exp\,(-\left(\frac{M_t - M^*}{\nu}\right)^2). \tag{4.9}$$

Pour ce modèle sans contrainte environnementale, nous disposons de résultats fournis par le programme GAMS de Franck Lecocq ; à chaque fois que cela est possible, les résultats

fournis par GAMS seront superposés à ceux donnés par SCILAB. Les résultats fournis par GAMS sont représentés en rouge, et ceux fournis par SCILAB seront représentés en bleu.

Nous allons étudier différentes versions du modèle STARTS *d'optimisation en coûts-avantages* ;
- avec différentes valeurs du paramètre d'inertie $\overline{\gamma}$;
- avec une fonction de dommage linéaire (4.8) ;
- avec une fonction de dommage exponentielle (4.9).

4.2.2 Cas où fonction de dommage est supposée linéaire

Nous allons supposer ici que la fonction de dommage s'écrit de la manière suivante :

$$d(M,t) = \kappa(t)\xi(M), \tag{4.10}$$

avec

$$\xi(M) = \theta(M - M_0), \tag{4.11}$$

où θ est un réel fixé et M_0 désigne la concentration en CO_2 à l'instant initial.

On rappelle que $\kappa(t)$ désigne le PIB du pays à l'instant t ; il est défini de la manière suivante :

$$\kappa(t) = 18\ 000 \exp 0.02\Delta t \times t. \tag{4.12}$$

Les valeurs des paramètres utilisées sont celle écrites dans le tableau (4.1.1), et nous fixons $\theta = 0.0000526$ unité par ppm, ce qui correspond pour l'état initial $M_0 = 360$ à un coût de dommage de l'ordre de un pour cent du PIB.

Pour des raisons de compatibilité avec les résultats founis par GAMS, nous n'avons pas introduit de coût final Φ car le programme de Franck Lecocq n'en prévoyait pas : il en résulte alors le fait que la commande a_{T-1} est nulle car le pays n'a pas intérêt à abattre au $(T-1)^{\text{e}}$ pas de temps, car nous ne prenons pas en compte dans le citère des coûts des émissions du T^{e} pas de temps.

Les résultats de la simulation sont représentés sur la figure 4.8.

Il apparaît que la fonction de dommage utilisée donne une trajectoire d'abattement qui croît doucement ; il semble que le coût des dommages ne soit pas assez significatif pour forcer le pays à réduire ses émissions de CO_2.

Nous allons donc multiplier θ par trois pour obtenir une trajectoire d'abattement plus significative : la fonction de dommage correspond maintenant à un coût de l'ordre de trois pour cent du PIB pour l'état initial $M_0 = 360$.

Les résultats sont représentés sur la figure 4.9.

Le coût des dommages étant plus élevé, la réponse est plus significative, avec cependant une pente à l'origine toujours très faible : *dans le cas de dommage linéaire, la pente de la fonction de dommage a peu d'impact sur les décisions à court terme.*

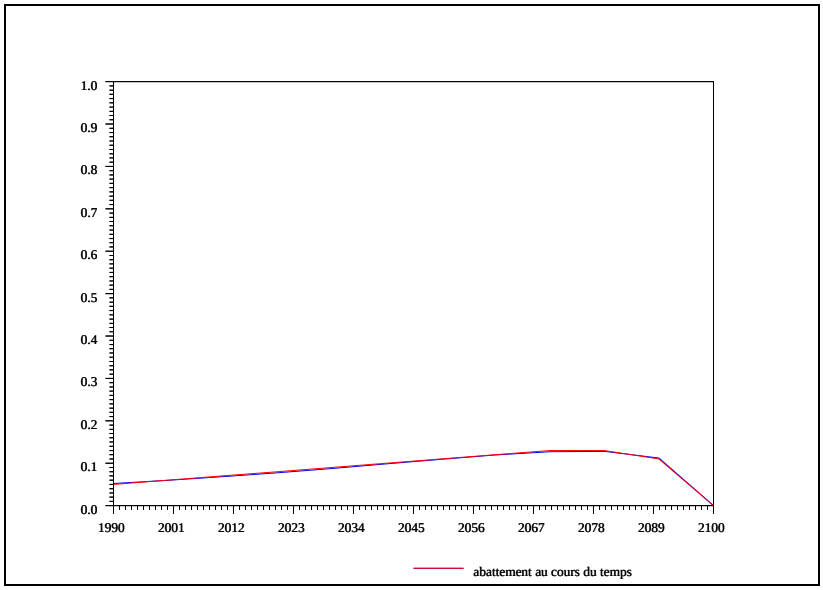

FIG. 4.8 – Trajectoires d'abattement, comparées entre GAMS (en rouge) et SCILAB(en bleu), avec une fonction de dommage linéaire correspondant à un coût de 1% du PIB à $t = 0$.

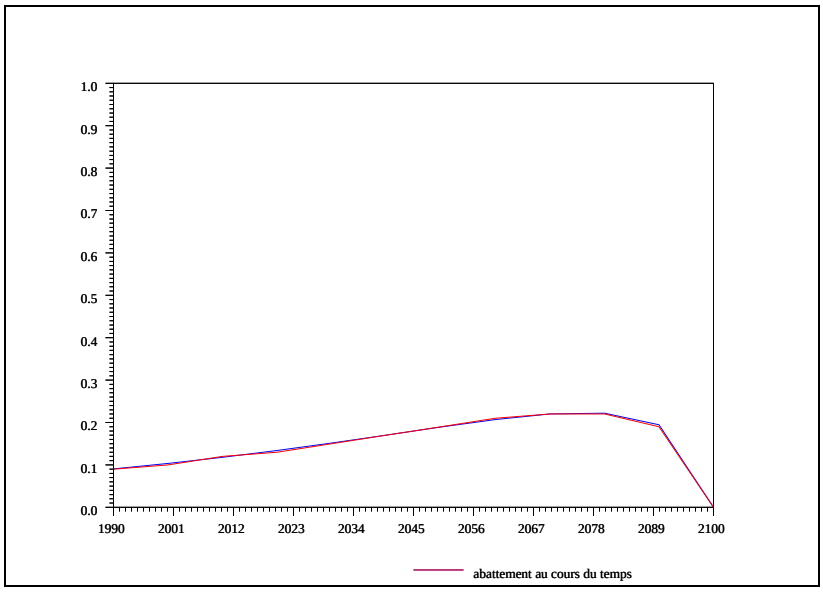

FIG. 4.9 – Trajectoires d'abattement, comparées entre GAMS (en rouge) et SCILAB(en bleu), avec une fonction de dommage linéaire correspondant à un coût de 3% du PIB à $t = 0$.

En complément, nous avons réalisé une simulation supplémentaire pour laquelle θ a été multiplié par dix. Cette simulation n'a pas été réalisée par GAMS, et comme il n'y a donc pas de comparaison à faire, le coût final ϕ a été introduit, ce qui donne la trajectoire représentée sur la figure (4.10).

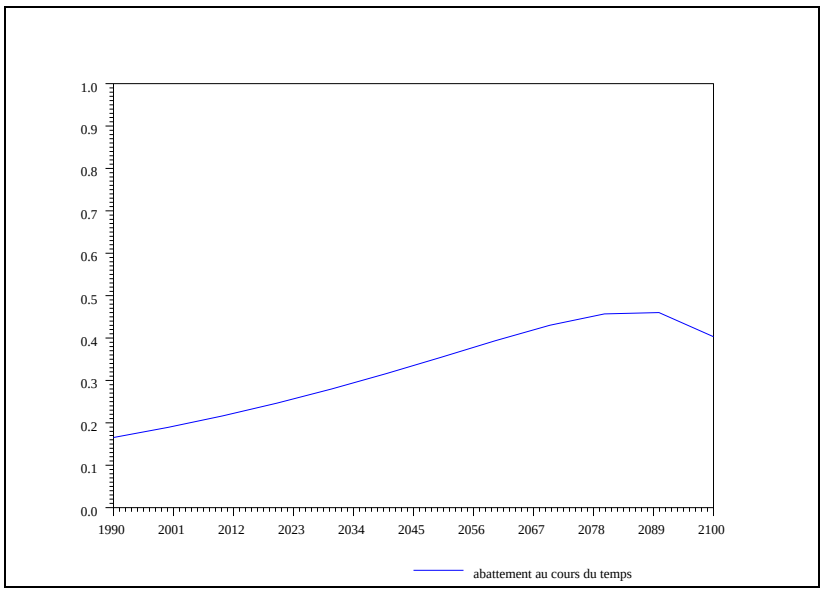

FIG. 4.10 – Trajectoire d'abattement , avec une fonction de dommage linéaire correspondant à un coût de 10% du PIB à $t = 0$.

.

4.2.3 Cas où la fonction de dommage est exponentielle

Nous allons maintenant étudier le cas où la fonction de dommage est exponentielle ; nous allons travailler avec le fonction suivante

$$d(M,t) = \kappa(t)\xi(M), \tag{4.13}$$

avec

$$\xi(M) = exp(-\left(\frac{(M - M^*)}{38.05}\right)^2),$$

où M^* est un paramètre réel à fixer.

Influence du paramètre $\overline{\gamma}$ sur les trajectoire d'abattement

Dans les simulations qui vont suivre, nous allons étudier l'influence du paramètre d'inertie $\overline{\gamma}$ sur les trajectoires d'abattement.

La concentration plafond M^* étant fixée à 560 ppm, nous allons faire varier le paramètre $\overline{\gamma}$: il prendra les valeurs 0.01 puis 0.005 et enfin 0.0025. Les résultats sont représentés sur les figures suivantes, avec toujours le même code couleur.

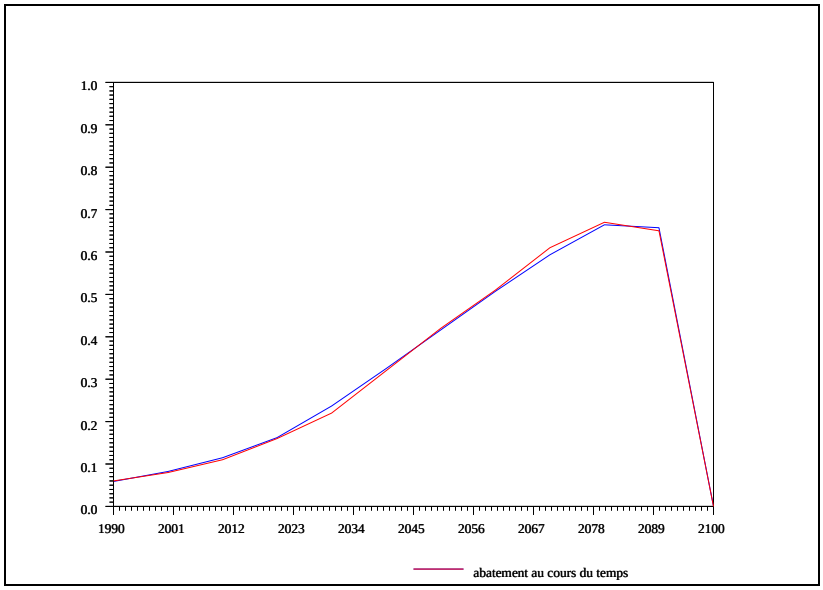

FIG. 4.11 – Trajectoire d'abattement avec fonction de dommage exponentielle, et faible inertie $\overline{\gamma} = 0.01$

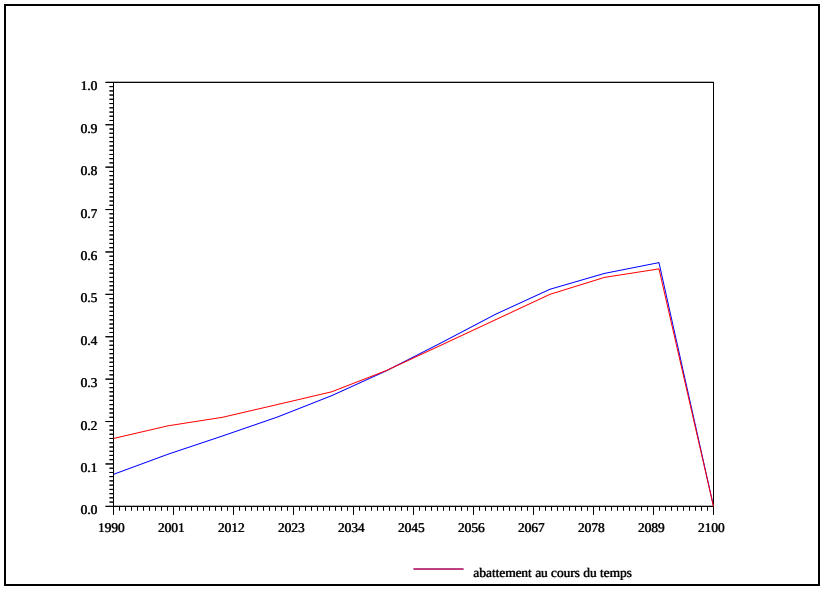

FIG. 4.12 – Trajectoire d'abattement avec fonction de dommage exponentielle, et $\overline{\gamma} = 0.005$

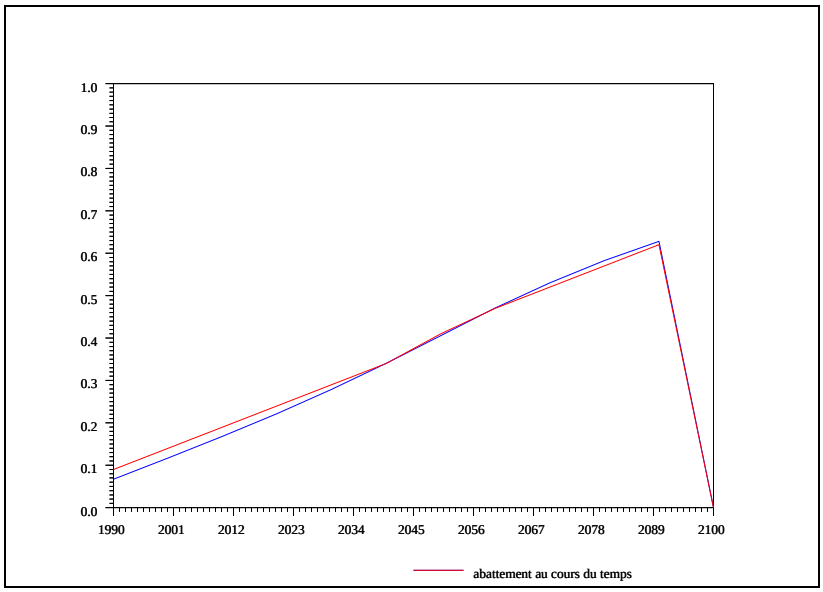

FIG. 4.13 – Trajectoire d'abattement avec fonction de dommage exponentielle, et forte inertie $\overline{\gamma} = 0.0025$

Influence du plafond de concentration M^* sur les trajectoires d'abattement

La fonction de dommage utilisée se comporte ainsi :
- tant que la concentration M est loin de la valeur limite M^*, elle est presque nulle ;
- dès que la concentration M se rapproche de la valeur M^*, elle prend des valeurs très grandes.

Tout se passe comme si nous avions introduit une contrainte de borne sur l'état

$$M < M^*,$$

et que nous pénaliserions cette contrainte dans le critère à minimiser. C'est pourquoi nous allons retrouver les mêmes types de trajectoires d'émissions que lors des simulations numériques des modèles STARTS *d'optimisation sous contrainte*. Dans les simulations qui vont suivre, nous allons examiner l'influence du paramètre M^* sur les trajectoires d'abattement.

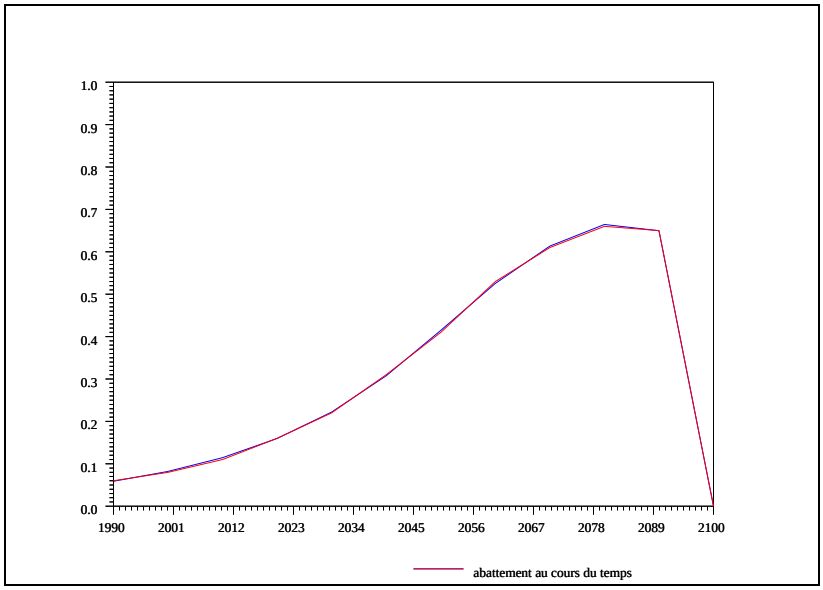

FIG. 4.14 – Trajectoire d'abattement avec fonction de dommage exponentielle et plafond $M^* = 560$

Les valeurs de M^* pour lesquelles nous avons des résultats de simulation sous GAMS sont un peu trop grande. En effet nous avons vu que, dans le cadre du modèle sous contrainte environnementale, la contrainte devient véritablement active pour $\overline{M} < 600$ ppm ; ici M^* joue le rôle de valeur seuil, et il apparaît que pour $M^* > 600$, la ¡¡ contrainte n'est plus active ¿¿, et les trajectoires d'abattement sont beaucoup moins significatives.

Cependant pour $M^* < 600$, nous obtenons des résultats similaires à ceux du modèle avec contrainte environnementale, comme en témoigne les figures 4.14, 4.17, et 4.18.

On notera que la fonction $\xi(M)$ utilisée présente certain inconvénient : elle est symétrique par rapport à l'axe $M = M^*$. Par exemple, pour $M^* = 500$, $\xi(400) = \xi(600)$, si bien que pour $M > M^*$, la fonction ξ est décroissante.

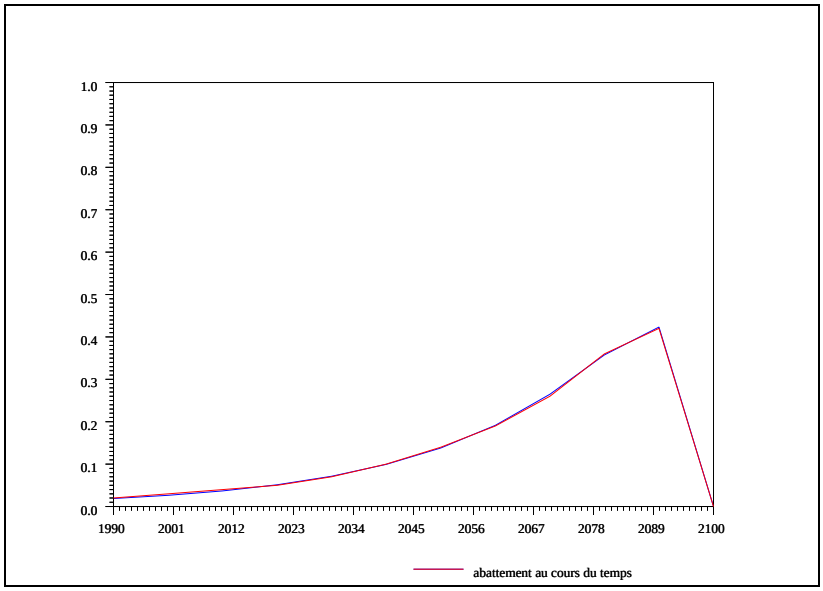

FIG. 4.15 – Trajectoire d'abattement et fonction de dommage exponentielle avec plafond $M^* = 655$

Nous sommes donc confrontés à des problèmes de bords, et cela explique pourquoi, sur la figure (4.18), la trajectoire est légèrement décroissante quand t s'approche de T. À l'extrême, nous pouvons même obtenir des comportements invraisemblables comme celui représenté sur la figure 4.16.

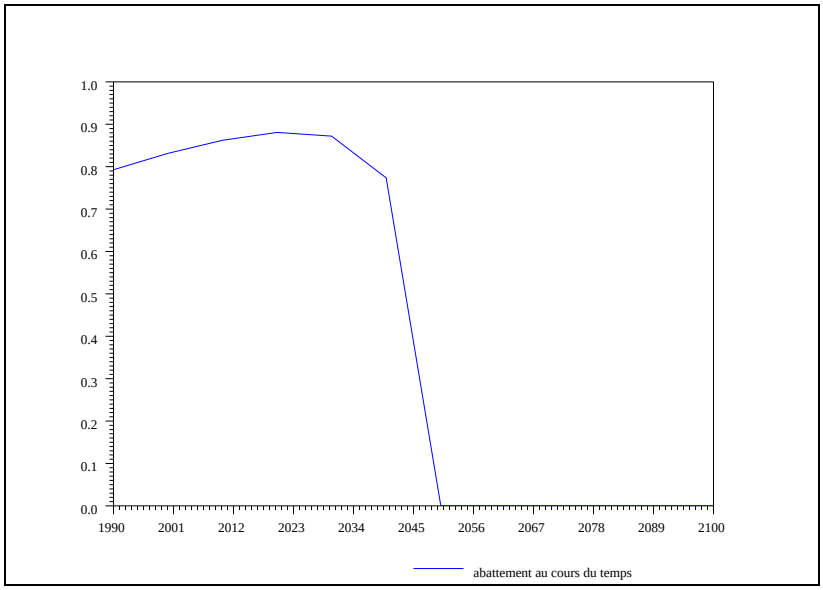

FIG. 4.16 – Trajectoire d'abattement avec fonction de dommage exponentielle et plafond $M^* = 400$

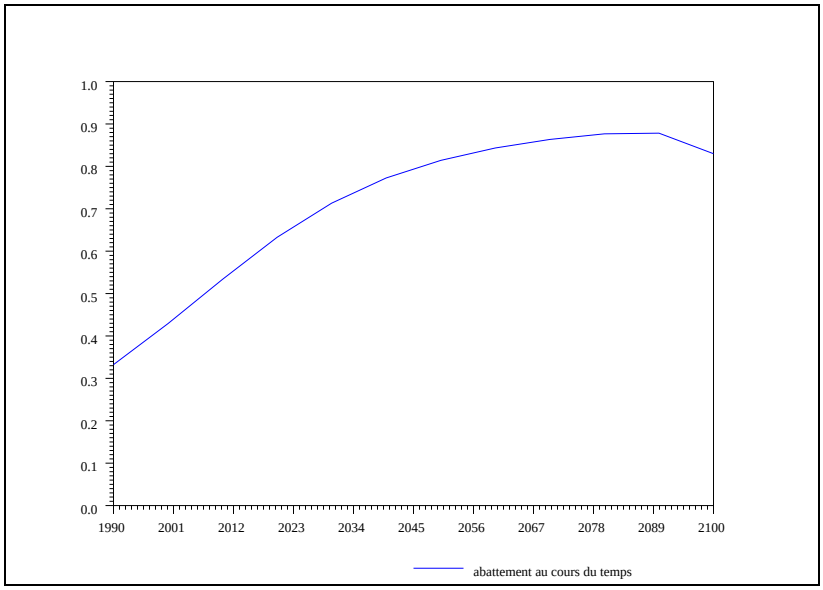

FIG. 4.17 – Trajectoire d'abattement avec fonction de dommage exponentielle et plafond $M^* = 450$

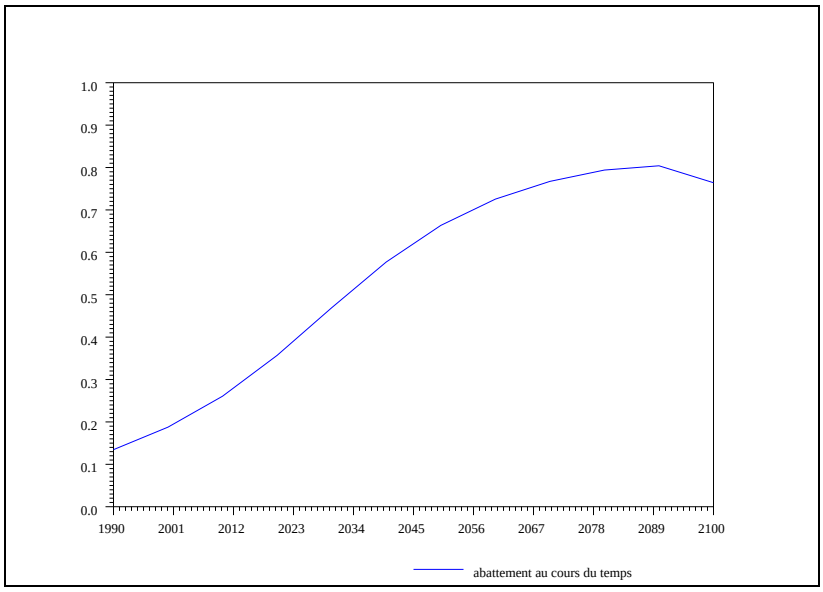

FIG. 4.18 – Trajectoire d'abattement avec fonction de dommage exponentielle et plafond $M^* = 500$

4.3 Synthèse sur la comparaison de SCILAB et de GAMS

Nous venons d'étudier une série de simulations numériques permettant de comparer les avantages respectifs des logiciels GAMS et SCILAB.

- Il apparaît, tout d'abord, que bien que les méthodes utilisées ne soient pas les mêmes, les résultats sont identiques dans la plupart des cas ;
- quand bien même les résultats sont différents, ceux fournis par SCILAB sont meilleurs au sens de la minimisation du critère $\mathcal{J}$ comme en témoigne le tableau 4.2.
- SCILAB offre une plus grande souplesse d'utilisation, avec des sorties graphiques immédiates, et des modifications de paramètre très aisées à mettre en œuvre.

Simulation	critère optimal pour *dynoptim*	critère optimal pour GAMS
optimisation sous contrainte avec $\overline{M} = 450$ ppm, et $\overline{\gamma} = 0.005$.	$\mathcal{J}_{opt} = 734.94808$	$\mathcal{J}_{gms} = 744.70864$
optimisation en coûts avantage, avec $M^* = 560$ ppm, et $\overline{\gamma} = 0.005$.	$\mathcal{J}_{opt} = 359.08836$	$\mathcal{J}_{gms} = 366.54334$

TAB. 4.2 – Comparaison des valeurs de critère à l'optimum fournies par GAMS et par SCILAB, dans les cas où il y avait une différence sensible entre les trajectoires d'abattement obtenues.

Conclusion

Ce stage a consisté à la mise sous forme mathématique les modèles économiques du CIRED. Cette mise en forme des problèmes servira par la suite pour les travaux de modélisation et d'expérimentation du CIRED qui seront réalisés avec SCILAB.

De plus, ce stage a permis une réflexion commune au CEREVE et au CIRED : les uns ont évalué les outils mathématiques nécessaires au traitement de problèmes économiques ; les autres ont réalisé l'importance de la clarification des problèmes posés en vue de leur résolution numérique.

Une grande partie de ce stage a été consacrée à la création d'outil numérique pour la résolution de problème d'optimisation dynamique. En effet le logiciel SCILAB ne disposait pas d'outils appropriés.

L'élaboration de ces outils a permis d'introduire des notions mathématiques complexes appliquées aux problèmes d'optimisation, telles, par exemple, les notions d'état adjoint, de dualité, de *Lagrangien augmenté*. En revanche cela a pris plus de temps que prévu, à cause de multiples tâtonnements et, par conséquent, l'élargissement du stage sur des problèmes d'économie plus complexes ne s'est pas réalisé.

Ainsi, ce stage a consisté beaucoup plus en l'étude de la simulation numérique de problèmes mathématiques, qu'en l'examen de problèmes économiques. Cependant, le simple fait de travailler pour le CIRED a permis de s'initier à la recherche en économie, et de se sensibiliser à des notions importantes en économie de l'environnement.

Finalement, ce stage a permis de découvrir le monde de la recherche à travers les interractions entre différents centres de recherche, mais aussi d'évaluer ce qu'était le véritable effort de recherche, avec ses obstacles, ses rebroussements, mais finalement des résultats.

Annexe A

Calcul de l'état adjoint

Ceci est la démonstration de la formule de calcul de l'état adjoint, utilisée pour calculer le gradient de la fonction $\mathcal{J}$ (2.13).

Nous supposons que

– la commande est de dimension p, *i.e* $u_t \in \mathbb{R}^p$;

– l'état est de dimension n, *i.e* $x_t \in \mathbb{R}^n$.

Nous voulons calculer le gradient de la fonction suivante :

$$\left\{ \begin{array}{rcl} J & = & \sum_{t=0}^{T-1} L(u_t, x_t, t) + \Phi(x_T) \\ \\ x_{t+1} & = & F(x_t, u_t, t) \\ x_0 & & \text{fixé.} \end{array} \right. \tag{A.1}$$

Lemme 1 *On considère le système discret suivant :*

$$\left\{ \begin{array}{rcl} dx_{k+1} & = & \alpha_k dx_k + \beta_k du_k \\ dx_0 & = & 0 \end{array} \right. \tag{A.2}$$

et la quantité G définie par :

$$G = \sum_{k=0}^{T-1} \delta_k dx_k + \delta_T dx_T \tag{A.3}$$

Alors, à condition de poser

$$\left\{ \begin{array}{rcl} y_{k-1} & = & \alpha'_k y_k + \delta'_k \\ y_{T-1} & = & \delta'_T, \end{array} \right. \tag{A.4}$$

on a aussi

$$G = \sum_{k=0}^{T-1} \left(y'_k \beta_k du_k \right). \tag{A.5}$$

Dans la suite $\frac{\partial f}{\partial x}$ désignera la matrice dont les éléments d'indices (i, j) sont $\frac{\partial f_i}{\partial x_j}$ Nous pouvons écrire [1]

$$\begin{array}{rcl} \Delta J & = & J(u + \delta u) - J(u) \\ & = & \sum_{t=0}^{T-1} \left(\frac{\partial L}{\partial x}(u_t, x_t, t)\delta x_t + \frac{\partial l}{\partial u}(u_t, x_t, t)\delta u_t \right) + \frac{d\Phi}{dx}(x_T, T)\delta x_T \end{array} \tag{A.6}$$

[1] Le symbole ' désigne ici la transposée.

avec

$$\left\{ \begin{array}{rcl} \delta x_{t+1} & = & \frac{\partial F}{\partial x}(x_t, u_t, t)\delta x_t + \frac{\partial F}{\partial u}(x_t, u_t)\delta u_t \\ \delta x_0 & = & 0 \end{array} \right. \tag{A.7}$$

Appliquons le lemme avec :

$$\left\{ \begin{array}{rcl} \Lambda_{t-1} & = & \left(\frac{\partial F}{\partial x}(x_t, u_t, t)\right)' \Lambda_t + \left(\frac{\partial L}{\partial x}(u_t, u_t, t)\right)' \\ \Lambda_{T-1} & = & \left(\frac{d\Phi}{dx}(x_T, T)\right)', \end{array} \right. \tag{A.8}$$

et alors nous obtenons

$$\Delta J = \sum_{t=0}^{T-1} \left(\Lambda_t' \frac{\partial F}{\partial u}(x_t, u_t, t) + \frac{\partial L}{\partial u}(u_t, x_t, t) \right) \times (\delta u)_t \tag{A.9}$$

ce qui nous donne l'expression voulue pour le gradient :

$$\frac{\partial J}{\partial u} = \sum_{t=0}^{T-1} \Lambda_t' \frac{\partial F}{\partial u}(x_t, u_t, t) + \frac{\partial L}{\partial u}(x_t, u_t, t) \tag{A.10}$$

démonstration du Lemme

$$\begin{array}{rcl} G & = & \sum_{k=1}^{T-1} (\delta_k dx_k) + \delta_T dx_T = \sum_{k=1}^{T-1} (y_{k-1} - \alpha_k' y_k)' dx_k + \delta_T' dx_T \\ & = & \sum_{k=1}^{T-1} (y_{k-1}' dx_k - y_k' \alpha_k dx_k) + \delta_T dx_T = \sum_{k=0}^{T-2} y_k' dx_{k+1} - \sum_{k=1}^{T-1} y_k' \alpha_k dx_k + \delta_T dx_T \\ & = & \underbrace{y_0' dx_1}_{y_0' \beta_0 du_0} + \sum_{k=1}^{T-2} y_k' \underbrace{(dx_{k+1} - \alpha_k dx_k)}_{\beta_k du_k} + \underbrace{-y_{T-1}' \alpha_{T-1} dx_{T-1} + \delta_T dx_T}_{y_{T-1}'(\beta_{T-1} du_{T-1})} \\ & = & \sum_{k=0}^{T-1} y_k' \beta_k du_k \end{array}$$

Annexe B

Le code

B.1 Les macros SCILAB

B.1.1 dynoptim

```
//-----------------------------------------------------------------
//                       optimisation dynamique
//  version sans contrainte
//          avec primitives+
//-----------------------------------------------------------------

//nom_... sont les noms des fonctions utiliser par le probleme qui consiste
// a trouver min J(u) avec J(u)=somme( L(u[t],x[t],t))+phi(u[T],x[T],T)
//           0<u<1               t=0..T-1
// en tenant compte de la dynamique :  x[t+1]=f(u[t],x[t],t)


//-------------------------------------------------------------------------


function
[uopt,eopt]=dynoptim(nom_L,nom_L_cmd,nom_L_etat,nom_f,nom_f_cmd,
nom_f_etat,nom_phi,nom_phi_etat,xb,xh,x0,e0)

// on fixe les fonctions a utiliser
//ATTENTION l'odre est le suivant :
doc_setf(nom_L,nom_L_cmd,nom_L_etat,nom_f,nom_f_cmd,
nom_f_etat,nom_phi,nom_phi_etat);


[fopt,uopt,gopt]=optim(dyn_ora,'b',xb,xh,x0,'ar',200,200);

[fopt,gopt,eopt]=doc_J(uopt,e0);
```

```
//-------------------------------------------------------------------------

// une fonction qui fait l'intermediaire entre J et optim
//  raison de compatibilite d' arguments et de retour


function [fo,go,ind]=dyn_ora(x,ind)
[fo,go,eo]=doc_J(x,e0);
```

B.1.2 dyoptimsc

```
//-----------------------------------------------------------------
//                        optimisation dynamique
//  version sans contrainte
//          avec primitives+
//-----------------------------------------------------------------

//nom_... sont les noms des fonctions utiliser par le probleme qui consiste
// a trouver min J(u) avec J(u)=somme( L(u[t],x[t],t))+phi(u[T],x[T],T)
//           0<u<1               t=0..T-1
// en tenant compte de la dynamique :  x[t+1]=f(u[t],x[t],t)


//-------------------------------------------------------------------------


function
[uopt,eopt]=dynoptimsc(nom_L,nom_L_cmd,nom_L_etat,nom_f,nom_f_cmd,
nom_f_etat,nom_phi,nom_phi_etat,xb,xh,x0,e0,einf,esup)

// on fixe les fonctions a utiliser
//ATTENTION l'odre est le suivant :
doc_setf_sc(nom_L,nom_L_cmd,nom_L_etat,nom_f,nom_f_cmd,
nom_f_etat,nom_phi,nom_phi_etat);



// definition des parametres du probleme
s_c=size(x0)
s_e=size(e0)
pas=s_c(2)
n_e=s_e(1)
```

```
k=zeros(n_e,1)+1;
p=zeros(2*n_e*(pas+1),1);
c=100;
r=50;


// fonction de minimisation de la fonction J situee
// dans le fichier cout_dyn.sce et programmee en c
//(fonctions.c,cout_dyn.c,interface.c)

xstart=x0;
maxiter=10;
stop=0.001;

for i=1:maxiter

[fopt,xopt,gopt]=optim(dyn_ora_sc,'b',xb,xh,xstart,'ar',200,200,imp=2);
t=doc_T_sc(xopt,e0,einf,esup,k);
p=(1-r/c)*p+r/c*maxi(0,p+c*t);


if i>1 then
if norm(xopt-xsav)<stop then
break
end;
end;

xsav=xopt;
xstart=xopt;

end;

uopt=xopt;

[fopt,gopt,eopt]=doc_J_sc(uopt,e0,einf,esup,k,c,p);


//-------------------------------------------------------------------------

// une fonction qui fait l'intermediaire entre J et optim
//  raison de compatibilite d' arguments et de retour


function [fo,go,ind]=dyn_ora_sc(x,ind)
[fo,go,lo]=doc_J_sc(x,e0,einf,esup,k,c,p);
```

B.2 Les modèles STARTS

B.2.1 Coût linéaire sans contrainte environnementale

```
//*****************************************************
//*                                                   *
//*          Description des fonctions necessaires    *
//*                                                   *
//*****************************************************




// definition du cout

E=[5.9623
,6.998
,8.4363
,9.9111
,11.018
,12.126
,13.233
,14.541
,15.848
,17.156
,18.463
,19.771];

alpha=1000; //cout de la backstop technology
nu=3;
theta=10*0.0000526;
M0=360;     // concentration initiale
Minf=274;   // concentration preindustrielle
rho=0.05;   // taux d'actualisation
sigma=0.01; // taux de disparition du CO2 dans les puits
delta=10;   // pas de temps
beta=0.38;  // taux d'absorption des emissions dans l'atmosphere
neta=38.05;
Ms=550;


deff('y=Kappa(t)','y=18000*exp(0.02*delta*t)');

deff('y=Lambda(t)','y=0.25+0.75*exp(-0.01*delta*t)');
```

```
//-------------------------- L et ses gradients-------------------------------

deff('y=L(a,x,t)','y=(alpha*E(t+1)*a^nu*Lambda(t)+
Kappa(t)*theta*(x(2)-M0))/(1+rho)^(delta*(t))');

deff('y=L_cmd(a,x,t)','y=alpha*E(t+1)*nu*a^(nu-1)*
Lambda(t)/(1+rho)^(delta*t)');

deff('y=L_etat(a,x,t)','y(1)=0,y(2)=Kappa(t)*theta/(1+rho)^(delta*t)');


// ------------------------f et ses gradients----------------------------------


deff('y=f(a,x,t)','y(1)=a,y(2)=x(2)+10*(beta*E(t+1)*(1-a)
-sigma*(x(2)-Minf))');

//         ATTENTION !
//
//      dfj
//      --- = res[j+N*i]
//      dxi

deff('y=f_etat(a,x,t)','y=zeros(2,2),y(1,1)=0,y(2,1)=0,
y(1,2)=0,y(2,2)=1-sigma*delta');

deff('y=f_cmd(a,x,t)','y=(1)=1,y(2)=-beta*E(t+1)*delta');



// ------------------------phi et ses gradients--------------------------------

deff('y=phi(x,t)','y=Kappa(t)*theta*(x(2)-M0)/(1+rho)^(delta*t)');

deff('y=phi_etat(x,t)','y(1)=0,
y(2)=Kappa(t)*theta/(1+rho)^(delta*t)');


//les resultats de Gams
Ugms1=[0.05
,0.06
,0.07
,0.08
,0.09
,0.10
,0.11
```

```
,0.12
,0.13
,0.13
,0.11
,0.00]

Ugms3=[0.09
,0.10
,0.12
,0.13
,0.15
,0.17
,0.19
,0.21
,0.22
,0.22
,0.19
,0.00]

// definition des parametres du probleme
e0=[0;360];
xb=zeros(1,12);
xh=zeros(1,12)+1;
x0=rand(1,12);


// oh que c'est joli
couleur=2;


[Uopt,Eopt]=dynoptim("L","L_cmd","L_etat","f","f_cmd","f_etat",
"phi","phi_etat",xb,xh,x0,e0);


T=[0:11];
T=1990+10*T;
plot2d(T,Uopt,couleur,"111","abatement au cours du temps ",
[1990,0,2100,1],[10,10,10,10]);
//plot2d(T,Ugms3',5,"111","abatement au cours du temps ",
[1990,0,2100,1],[10,10,10,10]);
```

B.2.2 Coût exponentiel avec inertie, et contrainte environnementale

```
xbasc(0);
```

```
//*****************************************************
//*                                                   *
//*          Description des fonctions necessaires    *
//*                                                   *
//*****************************************************



// definition du cout

E=[5.9623
,6.998
,8.4363
,9.9111
,11.018
,12.126
,13.233
,14.541
,15.848
,17.156
,18.463
,19.771];

alpha=1000; //cout de la backstop technology
nu=3;
theta=0.0000526;
M0=360;     // concentration initiale
Minf=274;   // concentration preindustrielle
rho=0.05;   // taux d'actualisation
sigma=0.01; // taux de disparition du CO2 dans les puits
delta=10;   // pas de temps
beta=0.38;  // taux d'absorption des emissions dans l'atmosphere
neta=38.05;
Ms=550;
e=0.0001
Gamma=0.005;


deff('y=Kappa(t)','y=18000*exp(0.02*delta*t)');

deff('y=Lambda(t)','y=0.25+0.75*exp(-0.01*delta*t)');

deff('y=xsi(m)','y=exp(-((m-Ms)/38.05)^2)');
deff('y=xsi_m(m)','y=-2*(m-Ms)/(38.05^2)*exp(-((m-Ms)/38.05)^2)');
```

```
//deff( 'y=gama(v)','y=1');
//deff( 'y=gama_v(v)','y=0');
deff( 'y=gama(v)','y=1/(2*Gamma*delta)*(v+sqrt(v*v+e))+1');
deff( 'y=gama_v(v)','y=1/(2*Gamma*delta)*(1+v/sqrt(v*v+e))');

deff('y=psi(u1,u2)','y=u2-u1-delta*Gamma');

deff('y=psi_u1(u1,u2)','y=-1')

deff('y=psi_u2(u1,u2)','y=1')

//-------------------------- L et ses gradients------------------------------

deff('y=L(a,x,t)','y=(alpha*E(t+1)*a^nu*Lambda(t)*
gama(psi(x(1),a)))/(1+rho)^(delta*t)');

deff('y=L_cmd(a,x,t)','y=alpha*E(t+1)*(nu*a^(nu-1)*Lambda(t)*
gama(psi(x(1),a))+a^nu*Lambda(t)*gama_v(psi(x(1),a)))/(1+rho)^(delta*t)');

deff('y=L_etat(a,x,t)','y(1)=-alpha*E(t+1)*a^nu*Lambda(t)*
gama_v(psi(x(1),a))/(1+rho)^(delta*t),y(2)=0');


// ------------------------f et ses gradients---------------------------------


deff('y=f(a,x,t)','y(1)=a,y(2)=x(2)+10*(beta*E(t+1)*(1-a)
-sigma*(x(2)-Minf))');

//         ATTENTION !
//
//      dfj
//      --- = res[j+N*i]
//      dxi

deff('y=f_etat(a,x,t)','y=zeros(2,2),y(1,1)=0,
y(2,1)=0,y(1,2)=0,y(2,2)=1-sigma*delta');

deff('y=f_cmd(a,x,t)','y(1)=1,y(2)=-beta*E(t+1)*delta');



// ------------------------phi et ses gradients-------------------------------

deff('y=phi(x,t)','y=0*Kappa(t)*xsi(x(2))/(1+rho)^(delta*t)');

deff('y=phi_etat(x,t)','y(1)=0,
```

```
y(2)=0*Kappa(t)*xsi_m(x(2))/(1+rho)^(delta*t)');

Ugms=[0.08
,0.11
,0.15
,0.24
,0.34
,0.44
,0.54
,0.64
,0.71
,0.73
,0.75];
//,0.00];

Ugms5=[0.15
,0.20
,0.25
,0.30
,0.36
,0.43
,0.51
,0.60
,0.68
,0.73
,0.75];


xopt  = [0.1015262
,0.1651512
,0.2211796
,0.2789328
,0.3460245
,0.4247461
,0.5156590
,0.6126594
,0.7003175
,0.7322702
,0.7463494];


// definition des parametres du probleme
e0=[0;360];
xb=zeros(1,11);
xh=zeros(1,11)+1;
x0=rand(1,11);
einf=[0;0];
```

```
esup=[1;450];

// oh que c'est joli
couleur=2;



[Uopt,Eopt]=dynoptimsc("L","L_cmd","L_etat","f","f_cmd","f_etat",
"phi","phi_etat",xb,xh,x0,e0,einf,esup);



T=[0:10];
T=1990+10*T;
plot2d(T,Uopt,couleur,"111","abattement au cours du temps ",
[1990,0,2100,1],[10,10,10,10]);
plot2d(T,Ugms5',5,"111","abattement au cours du temps ",
[1990,0,2100,1],[10,10,10,10]);

halt();


plot2d(T,Eopt(2,1:12),couleur,"011","pout",
[1990,360,2100,600],[10,10,10,10]);
```

B.3 Les primitives écrites en C

B.3.1 problème d'optimisation sans contrainte

B.3.2 eval_critère

```
#include <stdio.h>
#include <stdlib.h>
#include "utils.h"
#include "fonctions.h"
#include "cout_dyn.h"


/*****************************************************************
 *                       CALCUL DU COUT TOTAL                    *
 *                                                               *
 *                                ET                             *
 *                                                               *
 *                         CALCUL DU GRADIENT                    *
 *****************************************************************/
```

```
/*il s'agit de calculer le gradient d'une fonction s'ecrivant

  J(cmd)= somme  (L(cmd(t),etat(t),t)  + phi(etat(T))
         t=0..T-1

  en tenant compte de la dynamique:
           etat(0)=e0
   etat(t+1)=f(cmd(t),etat(t),t)

  les fonctions L, phi , f ainsi que leur gradient sont dans
  le fichier "fonctions.f"
*/

FILE* fid;

void J_GRAD(double* cmdt,int* n_pas,int* n_cmd,int* n_etat,
double* init_etat,double* etat,double* grd,double* FI,int last)
{

  /* les anciennes valeurs des tailles */
  static int o_lambda=0,o_gC_cmd=0,o_gf_cmd=0,o_gf_etat=0,o_l=0;

  /* les nouvelles */
  int T=*n_pas;  /* nombre de pas de temps */
  int N=*n_etat; /* taille de l'etat       */
  int C=*n_cmd;  /* taille de la commande  */


  static double* lambda;

  /* des intermediaires */
  double c; /*pour le calcul du cout*/
  static double* gf_etat=NULL;
  static double* gf_cmd=NULL;
  static double* gC_cmd=NULL;
  static double* l=NULL;
  int i,j,k;

  /* si c'est le dernier on nettoie */
  if(last==1)
    {
      net(&lambda);
      net(&gf_etat);
      net(&gf_cmd);
      net(&gC_cmd);
      net(&l);
```

```
    return ;
  }


/* sinon on fait de la place */
allocdouble(&lambda,N*(T+1),&o_lambda);
allocdouble(&gf_etat,N*N,&o_gf_etat);
allocdouble(&gf_cmd,N*C,&o_gf_cmd);
allocdouble(&gC_cmd,C,&o_gC_cmd);
allocdouble(&l,N,&o_l);







/*-------------------------------DEBUT-------------------------------*/


/* on calcul l'etat(N x (T+1)) en fonction des  T commandes
-------------------------------------------------------------*/

/*  on initialise l'etat*/

for(i=0;i<N;i++)
  etat[i]=init_etat[i];

/* on calcule etat(t+1) avec etat(t)*/

for(k=0;k<T;k++)
  {
    f(cmdt+C*k,etat+N*k,&k,etat+(k+1)*N,n_cmd,n_etat);

  }

/* on en profite pour calculer le cout total
-------------------------------------------*/

/* on initialise les variables de sortie */

*FI=0;

/* calcul de J */

for(k=0;k<T;k++)
```

```
    {
      L(cmdt+C*k,etat+N*k,&k,&c,n_cmd,n_etat);
      *FI+=c;

    }

  phi(etat+N*T,&T,&c,n_etat);
  *FI+=c;

  /*            on calcul l'etat adjoint (T x N+1)
en foction des T commandes et des T*(N+1) etats
   ----------------------------------------------------*/

  /* on initialise les lambdas */

  for(k=0;k<T+1;k++)
    for(i=0;i<N;i++)
      lambda[i+N*k]=0;

  /* on commence par lambda  en T (j=0...N-1) */

  phi_etat(etat+N*T,&T, lambda+N*T,n_etat);

  for(k=T-1;k>0;k--)
    {
      L_etat(cmdt+C*k,etat+N*k,&k,l,n_cmd,n_etat);
      f_etat(cmdt+C*k,etat+N*k,&k,gf_etat,n_cmd,n_etat);

      for(i=0;i<N;i++)
{
  for(j=0;j<N;j++)
    lambda[i+N*k]+=gf_etat[j+N*i]*lambda[j+N*(k+1)];
  lambda[i+N*k]+=l[i];
}
    }

  /*               Calcul du gradient du cout
   ---------------------------------------------------*/

  /* on initialise le gradient a zero*/

  for(k=0;k<T;k++)
    for(i=0;i<C;i++)
      grd[i+k*C]=0;

  for(k=0;k<T;k++)
    {
```

```
      f_cmd(cmdt+C*k,etat+N*k,&k,gf_cmd,n_cmd,n_etat);
      L_cmd(cmdt+C*k,etat+N*k,&k,gC_cmd,n_cmd,n_etat);
      for(i=0;i<C;i++)
{
  for(j=0;j<N;j++)
    grd[i+k*C]+=lambda[j+N*(k+1)]*gf_cmd[j+N*i];

  grd[i+k*C]+=gC_cmd[i];

}
    }

  /*                              FIN
  ---------------------------------------------------------------------*/


}
```

l'interface C-SCILAB

```
#include <string.h>
#include "stack-c.h"
#include "cout_dyn.h"
#include "doc_int.h"

/*************************************************
 * deal with errors in scilab functions
 *************************************************/

#include <setjmp.h>
static  jmp_buf Jcenv;

#define MySciString(ibegin,name,mlhs,mrhs) \
  if( ! C2F(scistring)(ibegin,name,mlhs,mrhs,strlen(name))) \
       { longjmp(Jcenv,-1);return 0; }

static int returned_from_longjump ;


/*********************************************************************
 *                        INTERFACE POUR SCILAB                      *
 *                                                                   *
 *                   [f,grf,lambda]=J(cmd,e0)                        *
 *********************************************************************/
```

```
int interfaceJ(char* fname)
{
  static int ierr;
  static int m1,n1,l1,m2,n2,l2; /* les variables d'entres*/
  static int m3,n3,l3,m4,n4,l4,m5,n5,l5;/* les outputs */
  static int minlhs=0, minrhs=2, maxlhs=3,maxrhs=2;

  CheckRhs(minrhs,maxrhs);
  CheckLhs(minlhs,maxlhs);

  /*          on recupere les variables entrees
  ------------------------------------------------*/
  GetRhsVar(1,"d",&m1,&n1,&l1);
  GetRhsVar(2,"d",&m2,&n2,&l2);


  /*  n1 est le nombre de pas de temps
      m1 est la taille de la commande
      m2 est la dimension de l'etat*/

  /*         on cree les variables de sorties
  ------------------------------------------------*/
  n3=1;
  m3=1;
  CreateVar(3,"d",&m3,&n3,&l3);

  n4=n1;
  m4=m1;
  CreateVar(4,"d",&m4,&n4,&l4);

  /* l'etat est de dimension N*(T+1)*/
  n5=(n1+1);
  m5=m2;
  CreateVar(5,"d",&m5,&n5,&l5);

  /* le dernier parametre sert a dire si c'est la derniere fois
     que l'on s'en sert*/

  if (( returned_from_longjump = setjmp(Jcenv)) != 0 )
    {
      Scierror(999,"%s: Internal error \r\n",fname);
      return 0;
    }

  J_GRAD(stk(l1),&n1,&m1,&m2,stk(l2),stk(l5),stk(l4),stk(l3),0);

  /*test(stk(l1),stk(l2),stk(l3),&m1,&m2);*/
```

```
  LhsVar(1)=3;
  LhsVar(2)=4;
  LhsVar(3)=5;

  return 0;
}


/*************************************************************************
 *                        INTERFACE POUR SCILAB                          *
 *                                                                       *
 *                          Setf(L,phi,f)                                *
 *************************************************************************/


/*  les noms des fonctions scilab sont dans les variables suivantes
-------------------------------------------------------------------------*/

char phi_name[nlgh+1],phi_etat_name[nlgh+1];
char f_name[nlgh+1],f_cmd_name[nlgh+1],f_etat_name[nlgh+1];
char L_name[nlgh+1],L_cmd_name[nlgh+1],L_etat_name[nlgh+1];



int interfaceSetf(char* fname)
{
  static int ierr;
  static int m1,n1,l1,m2,n2,l2,m3,n3,l3;/* les noms de L et de ses derivees  */
  static int m4,n4,l4,m5,n5,l5,m6,n6,l6;/* les noms de f et de ses derivees  */
  static int m7,n7,l7,m8,n8,l8;/* les noms de phi et de ses derivees*/

  static int minlhs=0, minrhs=8, maxlhs=1,maxrhs=8;

  CheckRhs(minrhs,maxrhs);
  CheckLhs(minlhs,maxlhs);

  /*          on recupere les variables entrees
  ------------------------------------------------*/
  GetRhsVar(1,"c",&m1,&n1,&l1); /* le nom de L        */
  GetRhsVar(2,"c",&m2,&n2,&l2); /* le nom de L_cmd    */
  GetRhsVar(3,"c",&m3,&n3,&l3); /* le nom de L_etat   */
  GetRhsVar(4,"c",&m4,&n4,&l4); /* le nom de f        */
  GetRhsVar(5,"c",&m5,&n5,&l5); /* le nom de f_cmd    */
  GetRhsVar(6,"c",&m6,&n6,&l6); /* le nom de f_etat   */
  GetRhsVar(7,"c",&m7,&n7,&l7); /* le nom de phi      */
  GetRhsVar(8,"c",&m8,&n8,&l8); /* le nom de phi_etat */
```

```
  strncpy(L_name,cstk(l1),nlgh);
  strncpy(L_cmd_name,cstk(l2),nlgh);
  strncpy(L_etat_name,cstk(l3),nlgh);
  strncpy(f_name,cstk(l4),nlgh);
  strncpy(f_cmd_name,cstk(l5),nlgh);
  strncpy(f_etat_name,cstk(l6),nlgh);
  strncpy(phi_name,cstk(l7),nlgh);
  strncpy(phi_etat_name,cstk(l8),nlgh);


  return 0;
}

/*************************************************************************
 *                        INTERFACE POUR SCILAB                          *
 *                                                                       *
 *                                  whasup()                             *
 *************************************************************************/

int interfacewashup(char* fname)
{

  static int ierr,un=1;
  static double p=1;
  static int minlhs=1, minrhs=0, maxlhs=1,maxrhs=0;

  CheckRhs(minrhs,maxrhs);
  CheckLhs(minlhs,maxlhs);


 /* le dernier parametre sert a dire que c'est la derniere fois
     que l'on s'en sert*/

  if (( returned_from_longjump = setjmp(Jcenv)) != 0 )
    {
      Scierror(999,"%s: Internal error \r\n",fname);
      return 0;
    }

  J_GRAD(&p,&un,&un,&un,&p,&p,&p,&p,1);

  return 0;

}
```

```
/************************** interface sciex ************************/

int sciex3(char fct[nlgh+1],double* cmd,double* etat,int* temps,
double* res,int* n_cmd,int* n_etat, int* n_res)
{

  // les variables utilisees sont les 6, 7 et 8
  static int l6,l7,l8;
  // on utilise trois arguments et on ne renvoie qu'un resultat
  static int ibegin=6,mlhs=1,mrhs=3;

  int i,un=1;

  CreateVar(6,"d",n_cmd,&un,&l6);
  for(i=0;i < *n_cmd;i++)
    stk(l6)[i]=cmd[i];

  CreateVar(7,"d",n_etat,&un,&l7);
  for(i=0;i < *n_etat;i++)
    stk(l7)[i]=etat[i];

  CreateVar(8,"d",&un,&un,&l8);
  stk(l8)[0]=*temps;

  /* execute la fonction fct */

  MySciString(&ibegin,fct,&mlhs,&mrhs);

  for(i=0;i<n_res[0];i++)
    res[i]=stk(l6)[i];

  return 0;
}

int sciex2(char fct[nlgh+1],double* etat,int* temps,
double* res,int* n_etat, int* n_res)
{

  // les variables utilisees sont les 6, 7
  static int l6,l7;
  // on utilise deux arguments et on ne renvoie qu'un resultat
  static int ibegin=6,mlhs=1,mrhs=2;

  int i,un=1;

  CreateVar(6,"d",n_etat,&un,&l6);
```

```
  for(i=0;i < *n_etat;i++)
    stk(l6)[i]=etat[i];

  CreateVar(7,"d",&un,&un,&l7);
  stk(l7)[0]=*temps;


  /* execute la fonction fct */

  MySciString(&ibegin,fct,&mlhs,&mrhs);

  for(i=0;i<n_res[0];i++)
    res[i]=stk(l6)[i];

  return 0;
}
```

le fichier de déclaration des fonctions

```
#include "fonctions.h"
#include "doc_int.h"

/*************************************************************************
 *                              fonction de cout                         *
 *************************************************************************/

void L(double* cmd,double* etat,int* temps,
double* res,int* n_cmd,int* n_etat)
{
  int R=1,pass;
  //on execute la macro L_name: resultat de taille 1
  pass=sciex3(L_name,cmd,etat,temps,res,n_cmd,n_etat,&R);
}


void L_cmd(double* cmd,double* etat,int* temps,
double* res,int* n_cmd,int* n_etat)
{
  int pass;
  //on execute la macro L_cmd_name: resultat de taille *n_cmd
  pass=sciex3(L_cmd_name,cmd,etat,temps,res,n_cmd,n_etat,n_cmd);

}
```

```
void L_etat(double* cmd,double* etat,int* temps,
double* res,int* n_cmd,int* n_etat)
{
  int pass;
  //on execute la macro L_etat_name: resultat de taille *n_etat
  pass=sciex3(L_etat_name,cmd,etat,temps,res,n_cmd,n_etat,n_etat);
}

/*****************************************************************
 *                 fonction de contraite finale                  *
 *****************************************************************/

void phi(double* etat,int* temps,double* res , int* n_etat)
{
  int R=1,pass;
  //on execute la macro L_name: resultat de taille 1
  pass=sciex2(phi_name,etat,temps,res,n_etat,&R);

}

void phi_etat(double* etat,int* temps,double* res , int* n_etat)
{
  int pass;
  //on execute la macro L_etat_name: resultat de taille *n_etat
  pass=sciex2(phi_etat_name,etat,temps,res,n_etat,n_etat);
}




/*****************************************************************
 *                              dynamique                        *
 *****************************************************************/



void f(double* cmd,double* etat,int* temps,
double* res,int* n_cmd,int* n_etat)

{
  int pass;
  //on execute la macro L_etat_name: resultat de taille *n_etat
  pass=sciex3(f_name,cmd,etat,temps,res,n_cmd,n_etat,n_etat);
}

void f_etat(double* cmd,double* etat,int* temps,
```

```
double* res,int* n_cmd,int* n_etat)
{
  int pass,R=*n_etat*(*n_etat);
  //on execute la macro L_etat_name: resultat de taille *n_etat*(*n_etat)
  pass=sciex3(f_etat_name,cmd,etat,temps,res,n_cmd,n_etat,&R);

}

void f_cmd(double* cmd,double* etat,int* temps,
double* res,int* n_cmd,int* n_etat)
{
  int pass,R=*n_etat*(*n_cmd);
  //on execute la macro f_cmd_name: resultat de taille *n_etat*(*n_cmd)
  pass=sciex3(f_cmd_name,cmd,etat,temps,res,n_cmd,n_etat,&R);
}
```

B.3.3 problème d'optimisation avec contraintes

eval_critère

```
#include <stdio.h>
#include <stdlib.h>
#include "utils.h"
#include "fonctions.h"
#include "cout_dyn.h"


/*********************************************************************
 *                        CALCUL DU COUT TOTAL                       *
 *                                                                   *
 *                                 ET                                *
 *                                                                   *
 *                           CALCUL DU GRADIENT                      *
 *********************************************************************/

/*il s'agit de calculer le gradient d'une fonction s'ecrivant

  J(cmd)= somme  (L(cmd(t),etat(t),t)  + phi(etat(T))
         t=0..T-1

  en tenant compte de la dynamique:
           etat(0)=e0
   etat(t+1)=f(cmd(t),etat(t),t)

  les fonctions L, phi , f ainsi que leur gradient sont dans
  le fichier "fonctions.f"
```

```
*/


extern int BeginVar;

void J_sc_GRAD(double* cmdt, int* n_pas, int* n_cmd, int* n_etat,
 double* init_etat, double* etatinf, double* etatsup,
 double *alpha, double c, double* p, double* etat,
 double* grd, double* FI, int last)
{

  /* les anciennes valeurs des tailles */
  static int o_lambda=0,o_gC_cmd=0,o_gf_cmd=0,o_gf_etat=0,o_l=0;

  /* les nouvelles */
  int T=*n_pas;  /* nombre de pas de temps */
  int N=*n_etat; /* taille de l'etat       */
  int C=*n_cmd;  /* taille de la commande  */


  static double* lambda;

  /* des intermediaires */
  double cout; /*pour le calcul du cout*/
  static double* gf_etat=NULL;
  static double* gf_cmd=NULL;
  static double* gC_cmd=NULL;
  static double* l=NULL;
  int i,j,k;

  /* si c'est le dernier on nettoie */
  if(last==1)
    {
      net(&lambda);
      net(&gf_etat);
      net(&gf_cmd);
      net(&gC_cmd);
      net(&l);

      return ;
    }

  /* sinon on fait de la place */
  allocdouble(&lambda,N*(T+1),&o_lambda);
  allocdouble(&gf_etat,N*N,&o_gf_etat);
  allocdouble(&gf_cmd,N*C,&o_gf_cmd);
  allocdouble(&gC_cmd,C,&o_gC_cmd);
```

```
  allocdouble(&l,N,&o_l);



  /* enfin on dit quelle est la derniere variable Scilab utilisee*/
  BeginVar=11;

  /*------------------------------DEBUT-------------------------------*/


  /* on calcul l'etat(N x (T+1)) en fonction des  T commandes
  ------------------------------------------------------------*/

  /*  on initialise l'etat*/

  for(i=0;i<N;i++)
    etat[i]=init_etat[i];

  /* on calcule etat(t+1) avec etat(t)*/

  for(k=0;k<T;k++)
    f_sc(cmdt+C*k,etat+N*k,&k,etat+(k+1)*N,n_cmd,n_etat);


  /* on en profite pour calculer le cout total
  -------------------------------------------*/

  /* on initialise les variables de sortie */

  *FI=0;

  /* calcul de J */

  for(k=0;k<T;k++)
    {
      L_sc(cmdt+C*k,etat+N*k,&k,etatinf,etatsup,alpha,c,p,&cout,n_cmd,n_etat);
      *FI+=cout;
    }

  phi_sc(etat+N*T,&T,etatinf,etatsup,alpha,c,p,&cout,n_etat);
  *FI+=cout;

  /*            on calcul l'etat adjoint (T x N+1)
en foction des T commandes et des T*(N+1) etats
   ------------------------------------------------*/

  /* on initialise les lambdas */
```

```
  for(k=0;k<T+1;k++)
    for(i=0;i<N;i++)
      lambda[i+N*k]=0;

  /* on commence par lambda  en T (j=0...N-1) */

  phi_sc_etat(etat+N*T,&T,etatinf,etatsup,alpha,c,p, lambda+N*T,n_etat);

  for(k=T-1;k>0;k--)
    {
      L_sc_etat(cmdt+C*k,etat+N*k,&k,etatinf,etatsup,alpha,c,p,l,n_cmd,n_etat);
      f_sc_etat(cmdt+C*k,etat+N*k,&k,gf_etat,n_cmd,n_etat);

      for(i=0;i<N;i++)
{
  for(j=0;j<N;j++)
    lambda[i+N*k]+=gf_etat[j+N*i]*lambda[j+N*(k+1)];
  lambda[i+N*k]+=l[i];
}
    }

  /*               Calcul du gradient du cout
   ------------------------------------------------*/

  /* on initialise le gradient a zero*/

  for(k=0;k<T;k++)
    for(i=0;i<C;i++)
      grd[i+k*C]=0;

  for(k=0;k<T;k++)
    {
      f_sc_cmd(cmdt+C*k,etat+N*k,&k,gf_cmd,n_cmd,n_etat);
      L_sc_cmd(cmdt+C*k,etat+N*k,&k,etatinf,etatsup,alpha,
c,p,gC_cmd,n_cmd,n_etat);
      for(i=0;i<C;i++)
{
  for(j=0;j<N;j++)
    grd[i+k*C]+=lambda[j+N*(k+1)]*gf_cmd[j+N*i];

  grd[i+k*C]+=gC_cmd[i];

}
    }

  /*                                FIN
```

```
  ------------------------------------------------------------------------*/


}



void THETA(double* cmdt, int* n_pas, int* n_cmd, int* n_etat,
double* init_etat, double* etatinf, double* etatsup, double *alpha,
  double* etat,double* Th, int last)
{

  int T=*n_pas;  /* nombre de pas de temps */
  int N=*n_etat; /* taille de l'etat       */
  int C=*n_cmd;  /* taille de la commande  */


  //les intermediaires
  int i,k;


  /* enfin on dit quelle est la derniere variable Scilab utilisee*/
  BeginVar=8;




  /* on calcule l'etat(N x (T+1)) en fonction des  T commandes
  ------------------------------------------------------------*/

  /*  on initialise l'etat*/

  for(i=0;i<N;i++)
    etat[i]=init_etat[i];

  /* on calcule etat(t+1) avec etat(t)*/

  for(k=0;k<T;k++)
    f_sc(cmdt+C*k,etat+N*k,&k,etat+(k+1)*N,n_cmd,n_etat);


  /* on calcule le vecteur contrainte theta(u)
     ----------------------------------------------*/

  //psup au dessus de pinf
```

```
  for(k=0;k<T+1;k++)
    for(i=0;i<N;i++)
      {
Th[i+2*N*k]=(etat[i+N*k]-etatsup[i])/alpha[i];
Th[i+N*(2*k+1)]=(etatinf[i]-etat[i+N*k])/alpha[i];
      }




}
```

B.3.4 l'interface C-SCILAB

```
#include <string.h>
#include "stack-c.h"
#include "cout_dyn.h"
#include "doc_int_sc.h"

/*************************************************
 * deal with errors in scilab functions
 *************************************************/

#include <setjmp.h>
static  jmp_buf Jcenv;

#define MySciString(ibegin,name,mlhs,mrhs) \
  if( ! C2F(scistring)(ibegin,name,mlhs,mrhs,strlen(name))) \
       { longjmp(Jcenv,-1);return 0; }

static int returned_from_longjump ;


/*********************************************************************
 *                        INTERFACE POUR SCILAB                      *
 *                                                                   *
 *               [f,grf,lambda]=J(cmd,e0,einf,esup,alpha,c,p)        *
 *********************************************************************/

int interfaceJ_sc(char* fname)
{
```

```
static int ierr;
/* les variables d'entres*/
static int m1,n1,l1,m2,n2,l2,m3,n3,l3,m4,n4,l4,m5,n5,l5,m6,n6,l6,m7,n7,l7;
/* les outputs */
static int m8,n8,l8,m9,n9,l9,m10,n10,l10;
static int minlhs=0, minrhs=7, maxlhs=3,maxrhs=7;

CheckRhs(minrhs,maxrhs);
CheckLhs(minlhs,maxlhs);

/*          on recupere les variables entrees
-------------------------------------------------*/
GetRhsVar(1,"d",&m1,&n1,&l1);
GetRhsVar(2,"d",&m2,&n2,&l2);
GetRhsVar(3,"d",&m3,&n3,&l3);//einf
GetRhsVar(4,"d",&m4,&n4,&l4);//esup
GetRhsVar(5,"d",&m5,&n5,&l5);//alpha
GetRhsVar(6,"d",&m6,&n6,&l6);//c
GetRhsVar(7,"d",&m7,&n7,&l7);//p
/*  n1 est le nombre de pas de temps
    m1 est la taille de la commande
    m2 est la dimension de l'etat*/

/*         on cree les variables de sorties
-------------------------------------------------*/
n8=1;
m8=1;
CreateVar(8,"d",&m8,&n8,&l8);

n9=n1;
m9=m1;
CreateVar(9,"d",&m9,&n9,&l9);

/* l'etat est de dimension N*(T+1)*/
n10=n1+1;
m10=m2;
CreateVar(10,"d",&m10,&n10,&l10);

/* le dernier parametre sert a dire si c'est la derniere fois
   que l'on s'en sert*/


if (( returned_from_longjump = setjmp(Jcenv)) != 0 )
  {
    Scierror(999,"%s: Internal error \r\n",fname);
    return 0;
  }
```

```
  J_sc_GRAD(stk(l1),&n1,&m1,&m2,stk(l2),stk(l3),stk(l4),stk(l5),
stk(l6)[0],stk(l7),stk(l10),stk(l9),stk(l8),0);

  /*test(stk(l1),stk(l2),stk(l3),&m1,&m2);*/

  LhsVar(1)=8;
  LhsVar(2)=9;
  LhsVar(3)=10;

  return 0;
}
```

```
/************************************************************************
 *                        INTERFACE POUR SCILAB                         *
 *                                                                      *
 *                          Setf(L,phi,f)                               *
 ************************************************************************/


/*  les noms des fonctions scilab sont dans les variables suivantes
------------------------------------------------------------------------*/

char phi_name[nlgh+1],phi_etat_name[nlgh+1];
char f_name[nlgh+1],f_cmd_name[nlgh+1],f_etat_name[nlgh+1];
char L_name[nlgh+1],L_cmd_name[nlgh+1],L_etat_name[nlgh+1];



int interfaceSetf_sc(char* fname)
{
  static int ierr;
  static int m1,n1,l1,m2,n2,l2,m3,n3,l3;/* les noms de L et de ses derivees  */
  static int m4,n4,l4,m5,n5,l5,m6,n6,l6;/* les noms de f et de ses derivees  */
  static int m7,n7,l7,m8,n8,l8;/* les noms de phi et de ses derivees*/

  static int minlhs=0, minrhs=8, maxlhs=1,maxrhs=8;

  CheckRhs(minrhs,maxrhs);
```

```
  CheckLhs(minlhs,maxlhs);

  /*          on recupere les variables entrees
  ------------------------------------------------*/
  GetRhsVar(1,"c",&m1,&n1,&l1); /* le nom de L        */
  GetRhsVar(2,"c",&m2,&n2,&l2); /* le nom de L_cmd    */
  GetRhsVar(3,"c",&m3,&n3,&l3); /* le nom de L_etat   */
  GetRhsVar(4,"c",&m4,&n4,&l4); /* le nom de f        */
  GetRhsVar(5,"c",&m5,&n5,&l5); /* le nom de f_cmd    */
  GetRhsVar(6,"c",&m6,&n6,&l6); /* le nom de f_etat   */
  GetRhsVar(7,"c",&m7,&n7,&l7); /* le nom de phi      */
  GetRhsVar(8,"c",&m8,&n8,&l8); /* le nom de phi_etat */


  strncpy(L_name,cstk(l1),nlgh);
  strncpy(L_cmd_name,cstk(l2),nlgh);
  strncpy(L_etat_name,cstk(l3),nlgh);
  strncpy(f_name,cstk(l4),nlgh);
  strncpy(f_cmd_name,cstk(l5),nlgh);
  strncpy(f_etat_name,cstk(l6),nlgh);
  strncpy(phi_name,cstk(l7),nlgh);
  strncpy(phi_etat_name,cstk(l8),nlgh);


  return 0;
}



/*************************************************************************
 *                        INTERFACE POUR SCILAB                          *
 *                                                                       *
 *                   [theta,etat]=T(cmd,e0,einf,esup,alpha)              *
 *************************************************************************/

int interfaceT_sc(char* fname)
{
  static int ierr;
  /* les variables d'entres*/
  static int m1,n1,l1,m2,n2,l2,m3,n3,l3,m4,n4,l4,m5,n5,l5;
  /* les outputs */
  static int m6,n6,l6,m7,n7,l7;
  static int minlhs=0, minrhs=5, maxlhs=2,maxrhs=5;

  CheckRhs(minrhs,maxrhs);
  CheckLhs(minlhs,maxlhs);
```

```
  /*          on recupere les variables entrees
  ------------------------------------------------*/
  GetRhsVar(1,"d",&m1,&n1,&l1);
  GetRhsVar(2,"d",&m2,&n2,&l2);
  GetRhsVar(3,"d",&m3,&n3,&l3);//einf
  GetRhsVar(4,"d",&m4,&n4,&l4);//esup
  GetRhsVar(5,"d",&m5,&n5,&l5);//alpha

  /*  n1 est le nombre de pas de temps
      m1 est la taille de la commande
      m2 est la dimension de l'etat*/

  /*         on cree les variables de sorties
  ------------------------------------------------*/
  n6=1;
  m6=2*m2*(n1+1);
  CreateVar(6,"d",&m6,&n6,&l6);

  /* l'etat est de dimension N*(T+1)*/
  n7=(n1+1);
  m7=m2;
  CreateVar(7,"d",&m7,&n7,&l7);

  /* le dernier parametre sert a dire si c'est la derniere fois
     que l'on s'en sert*/


  if (( returned_from_longjump = setjmp(Jcenv)) != 0 )
    {
      Scierror(999,"%s: Internal error \r\n",fname);
      return 0;
    }

  THETA(stk(l1),&n1,&m1,&m2,stk(l2),stk(l3),stk(l4),stk(l5),stk(l7),stk(l6),0);

  LhsVar(1)=6;
  LhsVar(2)=7;
  //fprintf(fid,"%s \n","fin du calcul");
  return 0;
}



/*************************************************************************
 *                       INTERFACE POUR SCILAB                           *
 *                                                                       *
 *                            washup()                                   *
```

```
 ***********************************************************************/



int interfacewashup_sc(char* fname)
{

  static int ierr,un=1;
  static double p=1;
  static int minlhs=1, minrhs=0, maxlhs=1,maxrhs=0;

  CheckRhs(minrhs,maxrhs);
  CheckLhs(minlhs,maxlhs);

  /* le dernier parametre sert a dire que c'est la derniere fois
     que l'on s'en sert*/

  if (( returned_from_longjump = setjmp(Jcenv)) != 0 )
    {
      Scierror(999,"%s: Internal error \r\n",fname);
      return 0;
    }

  J_sc_GRAD(&p,&un,&un,&un,&p,&p,&p,&p,p,&p,&p,&p,&p,1);
  return 0;

}




/************************** interface sciex ************************/




int sciex3_sc(char fct[nlgh+1],int begin,double* cmd,double* etat,
int* temps,double* res,int* n_cmd,int* n_etat, int* n_res)
{

  // les variables utilisees sont les 11, 12 et 13
  static int l1,l2,l3;
  // on utilise trois arguments et on ne renvoie qu'un resultat
```

```
  static int ibegin,mlhs=1,mrhs=3;

  int i,un=1;

  ibegin=begin;
  CreateVar(ibegin,"d",n_cmd,&un,&l1);
  for(i=0;i < *n_cmd;i++)
    {
      stk(l1)[i]=cmd[i];

    }
  CreateVar(ibegin+1,"d",n_etat,&un,&l2);
  for(i=0;i < *n_etat;i++)
    {
      stk(l2)[i]=etat[i];

    }

  CreateVar(ibegin+2,"d",&un,&un,&l3);
  stk(l3)[0]=*temps;


  /* execute la fonction fct */

  MySciString(&ibegin,fct,&mlhs,&mrhs);

  for(i=0;i<n_res[0];i++)
    {
      res[i]=stk(l1)[i];

    }
  return 0;
}

int sciex2_sc(char fct[nlgh+1],int begin,double* etat,int* temps,
double* res,int* n_etat, int* n_res)
{

  // les variables utilisees sont les 11 et 12
  static int l1,l2;
  // on utilise trois arguments et on ne renvoie qu'un resultat
  static int ibegin,mlhs=1,mrhs=2;

  int i,un=1;

  ibegin=begin;
  CreateVar(ibegin,"d",n_etat,&un,&l1);
```

```
  for(i=0;i < *n_etat;i++)
    {
      stk(l1)[i]=etat[i];

    }
  CreateVar(ibegin+1,"d",&un,&un,&l2);
  stk(l2)[0]=temps[0];


  /* execute la fonction fct */

  MySciString(&ibegin,fct,&mlhs,&mrhs);
  for(i=0;i<n_res[0];i++)
    {
      res[i]=stk(l1)[i];

    }

  return 0;
}
```

le fichier de déclaration des fonctions

```
#include <math.h>
#include "fonctions.h"
#include "stack-c.h"
#include "doc_int_sc.h"


/*************************************************************************
 *                              fonction de cout                         *
 *************************************************************************/



void L_sc(double* cmd,double* etat,int* temps,double* etatinf,
double* etatsup,double* alpha,double c,double* p,
double* res,int* n_cmd,int* n_etat)
{
  int R=1,pass,i=0;
  int N=*n_etat;
  //on execute la macro L_name: resultat de taille 1
  pass=sciex3_sc(L_name,BeginVar,cmd,etat,temps,res,n_cmd,n_etat,&R);

  //rajout pour Lagrangien augmenté
  for(i=0;i<n_etat[0];i++)
```

```
    {
      if((etat[i]-etatsup[i])/alpha[i]+p[i]/c>0)
*res+=c/2*pow((etat[i]-etatsup[i])/alpha[i]+p[i]/c,2);
      if((etatinf[i]-etat[i])/alpha[i]+p[i+N]/c>0)
*res+=c/2*pow((etatinf[i]-etat[i])/alpha[i]+p[i+N]/c,2);
    }
}


void L_sc_cmd(double* cmd,double* etat,int* temps,double* etatinf,
double* etatsup,double* alpha,double c,double* p,
double* res,int* n_cmd,int* n_etat)
{
  int pass;
  //on execute la macro L_cmd_name: resultat de taille *n_cmd
  pass=sciex3_sc(L_cmd_name,BeginVar,cmd,etat,temps,res,n_cmd,n_etat,n_cmd);

}

void L_sc_etat(double* cmd,double* etat,int* temps,double* etatinf,
double* etatsup,double* alpha,double c,double* p,
double* res,int* n_cmd,int* n_etat)
{
  int pass,i=0;
  int N=*n_etat;
  //on execute la macro L_etat_name: resultat de taille *n_etat
  pass=sciex3_sc(L_etat_name,BeginVar,cmd,etat,temps,res,n_cmd,n_etat,n_etat);

  //rajout pour Lagrangien augmenté

  for(i=0;i<n_etat[0];i++)
    {
      if((etat[i]-etatsup[i])/alpha[i]+p[i]/c>0)
res[i]+=c*((etat[i]-etatsup[i])/alpha[i]+p[i]/c)/alpha[i];
      if((etatinf[i]-etat[i])/alpha[i]+p[i+N]/c>0)
*res+=-c*((etatinf[i]-etat[i])/alpha[i]+p[i+N]/c)/alpha[i];
    }
}



/*****************************************************************
 *                fonction de contraite finale                   *
 *****************************************************************/
```

```
void phi_sc(double* etat,int* temps,double* etatinf,double* etatsup,
double* alpha,double c,double* p,double* res,int* n_etat)
{
  int R=1,pass,i=0;
  int N=*n_etat;
  //on execute la macro L_name: resultat de taille 1
  pass=sciex2_sc(phi_name,BeginVar,etat,temps,res,n_etat,&R);

  // rajout pour le lagrangien augmente

  for(i=0;i<n_etat[0];i++)
    {
      if((etat[i]-etatsup[i])/alpha[i]+p[i]/c>0)
*res+=c/2*pow((etat[i]-etatsup[i])/alpha[i]+p[i]/c,2);
      if((etatinf[i]-etat[i])/alpha[i]+p[i+N]/c>0)
*res+=c/2*pow((etatinf[i]-etat[i])/alpha[i]+p[i+N]/c,2);
    }

}

void phi_sc_etat(double* etat,int* temps,double* etatinf,double* etatsup,
double* alpha,double c,double* p,double* res,int* n_etat)
{
  int pass,i=0;
  int N=*n_etat;
  //on execute la macro L_etat_name: resultat de taille *n_etat
  pass=sciex2_sc(phi_etat_name,BeginVar,etat,temps,res,n_etat,n_etat);

  for(i=0;i<n_etat[0];i++)
    {
      if((etat[i]-etatsup[i])/alpha[i]+p[i]/c>0)
res[i]+=c*((etat[i]-etatsup[i])/alpha[i]+p[i]/c)/alpha[i];

      if((etatinf[i]-etat[i])/alpha[i]+p[i+N]/c>0)
*res+=-c*((etatinf[i]-etat[i])/alpha[i]+p[i+N]/c)/alpha[i];
    }
}




/*********************************************************************
 *                              dynamique                            *
 *********************************************************************/
```

```
void f_sc(double* cmd,double* etat,int* temps,double* res,
int* n_cmd,int* n_etat)

{
  int pass;
  //on execute la macro L_etat_name: resultat de taille *n_etat

  pass=sciex3_sc(f_name,BeginVar,cmd,etat,temps,res,n_cmd,n_etat,n_etat);
}

void f_sc_etat(double* cmd,double* etat,int* temps,double* res,
int* n_cmd,int* n_etat)
{
  int pass,R=*n_etat*(*n_etat);
  //on execute la macro L_etat_name: resultat de taille *n_etat*(*n_etat)
  pass=sciex3_sc(f_etat_name,BeginVar,cmd,etat,temps,res,n_cmd,n_etat,&R);

}

void f_sc_cmd(double* cmd,double* etat,int* temps,double* res,
int* n_cmd,int* n_etat)
{
  int pass,R=*n_etat*(*n_cmd);
  //on execute la macro f_cmd_name: resultat de taille *n_etat*(*n_cmd)
  pass=sciex3_sc(f_cmd_name,BeginVar,cmd,etat,temps,res,n_cmd,n_etat,&R);
}
```

Bibliographie

Cohen, Guy (1999). Cours de convexité et optimisation.

Culioli, Jean Christophe (1994). *Introduction à l'Optimisation*.