Algorithmes numériques en commande: état de l'art et perspectives

Juan Carlos Zúñiga[1] — Didier Henrion[2,3]

[1] *Département de Mathématiques, Université de Guadalajara*
Marcelino García Barragán 1421, c.p. 44430, Guadalajara, Mexique.

[2] *LAAS-CNRS. 7 Avenue du Colonel Roche, 31077 Toulouse, France.*

[3] *Département de Théorie de la Commande, Faculté de Génie Electrique, Université Technique Tchèque à Prague, Technická 2, 16627 Prague, République Tchèque.*

e-mail: juan.zuniga@red.cucei.udg.mx, henrion@laas.fr

RÉSUMÉ. *On présente un état de l'art des algorithmes numériques en commande. On décrit les deux grandes approches à la modélisation et l'analyse des systèmes linéaires: l'approche espace d'état et l'approche polynomiale. D'une part, on montre comment l'algèbre linéaire numérique, discipline très bien développée et pilier du calcul scientifique, a permis de résoudre des problèmes compliqués en commande avec l'approche espace d'état. D'autre part, on souligne l'importance de consolider une algèbre polynomiale numérique pour résoudre les problèmes numériques présents dans l'approche polynomiale. Les deux premièrs chapitres de cet article sont des introductions aux algorithmes numériques et au calcul scientifique en général. Les deux chapitres suivants se focalisent sur l'approche espace d'état et l'approche polynomiale en commande.*

ABSTRACT. *We present a survey on numerical algorithms in control. We describe the two major approaches to modeling and analysis of linear systems: the state-space approach and the polynomial approach. On the one hand, we show how the very well developed numerical linear algebra, basis of scientific computing, has allowed to solve complicated control problems with the state-space approach. On the other hand, we emphasize the relevance of consolidating a numerical polynomial algebra to solve the numerical problems in the polynomial approach. The first two chapters of this paper are introductions to numerical algorithms and scientific computing in general. The next two chapters focus on the state-space approach and the polynomial approach to systems control.*

MOTS-CLÉS : *Théorie de la commande, algébre linéaire numérique, matrices polynomiales, logiciels de commande assistée par ordinateur.*

1re soumission à *RS - JESA*, le 21 juin 2006

KEYWORDS: *Control theory, numerical linear algebra, polynomial matrices, computer-aided control system design software.*

1. La précision finie et les problèmes numériques

L'exécution ordonnée d'un ensemble de pas (instructions ou opérations) vers la solution d'un problème quelconque est quelque chose que l'homme a fait depuis la nuit des temps, même avant qu'il existe le mot algorithme pour définir ce processus (Chabert, 1999). Le mot algorithme garde une connotation mathématique qui convient parfaitement étant donnée la thématique de ce travail. Cependant, d'autres possibles synonymes comme processus, méthode, technique, procédure, règle ou recette mettent en évidence que tout le monde applique des algorithmes même inconsciemment.

Le mot algorithme est dérivé de l'arabe Al-Khwārizmī qui était le nom du mathématicien auteur du plus ancien travail connu d'algèbre. Dans son travail, Al-Khwārizmī soulignait l'importance d'appliquer une procédure méthodique pour résoudre des problèmes comme certains types d'équations quadratiques. Ce travail et beaucoup d'autres, bases de ce que l'on appelle l'algèbre, sont traduits en latin au 12ème siècle. À ce moment, et à cause de l'introduction du nouveau système de numérotation, on utilisait le mot algorithme pour nommer certaines procédures arithmétiques. Quand la numération arabe a été finalement adoptée, appliquer un algorithme signifiait simplement faire de l'algèbre. Avec le temps, l'algèbre est devenue une discipline beaucoup plus compliquée que la solution d'équations quadratiques et les algorithmes sont devenus simplement des outils pour résoudre des problèmes spécifiques de manière méthodique, systématique.

Actuellement la liste des algorithmes qui servent à résoudre un problème donné peut être, pour certains problèmes, très vaste. Les algorithmes ont évolué. Une tendance naturelle est de les rendre à chaque fois plus généraux. Cette tendance est aussi encouragée par le développement de la science qui impose, à chaque fois, des problèmes plus compliqués. Prenons par exemple la solution d'un système d'équations $Ax = b$. En 1750 Cramer a formulé une procédure générale pour résoudre ce système. Avec la règle de Cramer, le problème paraissait résolu. Cependant, quand on a voulu appliquer cette règle aux nouveaux problèmes en astronomie de cette époque, où le nombre d'équations était grand, le nombre de multiplications qu'on devait faire était énorme[1]. Face à ce problème, d'autres algorithmes, qui réduisaient le nombre d'opérations, ont été développés. C'est le cas, par exemple, de l'élimination Gaussienne en 1810.

Les algorithmes n'ont pas seulement évolué grâce aux problèmes posés par la science, mais aussi la science a avancé grâce à l'étude, l'analyse et l'application des algorithmes. Par exemple, ce n'est qu'en 1815 que Cauchy a démontré la règle de Cramer pour la première fois, en donnant l'origine à l'étude des déterminants. De manière similaire, c'est en 1850 que Sylvester a introduit le terme "matrice", des années après que ses propriétés fondamentales aient été établies (Chabert, 1999).

Avec le développement des ordinateurs dans les dernières décennies, l'étude et l'analyse des algorithmes ont pris de nouveau beaucoup d'importance. Malgré la ca-

1. Environ 300 millions de multiplications pour 10 équations (Chabert, 1999).

pacité de calcul que les ordinateurs offrent, la qualité de la solution obtenue n'est pas toujours ce que l'on peut espérer. En particulier, on peut vérifier que la plupart des algorithmes de facile exécution à la main et qui donnent de bons résultats ne deviennent plus fiables si on les codifie dans l'ordinateur. La cause : la précision finie présente dans tous les systèmes numériques. Cette précision finie limite la quantitéé de nombres réels que l'ordinateur peut représenter, et par conséquent des erreurs dues à l'arrondi sont introduites (Wilkinson, 1994).

Exemple 1 *La solution de l'opération* $0.3 - 0.2 - 0.1$ *calculée par Matlab*[2] (Mathworks, 2004) *en double précision*[3] *est* $-2.77... \times 10^{-17} \neq 0$. *La cause est simplement qu'aucun des nombres qui intervient dans le reste ne peut être représenté exactement par l'ordinateur. Cet exemple extrait de* (Boettcher, 2000) *montre que les erreurs d'arrondi apparaissent dès les problèmes les plus simples.*

Avec la précision finie, quelques propriétés des opérations arithmétiques basiques comme la somme ou la multiplication ne sont pas vérifiées. Tel est le cas de la propriété associative de la somme comme démontré dans l'exemple suivant.

Exemple 2 *Considérons un schéma arithmétique à double précision. L'opération* $1 + 10^{20} - 10^{20}$ *peut être effectuée par l'ordinateur comme* $1 + (10^{20} - 10^{20})$ *qui donne le résultat correct, ou bien comme* $(1 + 10^{20}) - 10^{20}$ *qui donne* 0 *car les nombres* $1 + 10^{20}$ *et* 10^{20} *sont représentés de la même manière par l'ordinateur avec la précision donnée.*

Intuitivement on peut deviner quelle est la façon correcte d'effectuer l'opération. Même quand il y a plusieurs opérations, on peut décider la précision avec laquelle chaque opération est effectuée en prédisant l'effet de chaque erreur dans le résultat final.

Exemple 3 *On veut calculer les racines* $x = \alpha$ *et* $x = \beta$ *de l'équation quadratique* $p(x) = ax^2 + bx + c$. *À l'aide de la formule classique qu'on apprend à l'école, on peut vérifier que*

$$\alpha = \frac{-b + d}{2a}, \qquad \beta = \frac{-b - d}{2a}$$

où $d = \sqrt{b^2 - 4ac}$. *Supposons que* $b > 0$ *est tel que* $b^2 \gg 4ac$. *Ainsi, au moment de calculer* β, *on peut considérer que* $d \approx b$ *et choisir* $\hat{\beta} = -b/a$ *comme une bonne approximation de la valeur réelle de* β. *Par contre, au moment de calculer* α *on a intérêt à obtenir* d *avec le plus grand nombre de chiffres significatifs de telle sorte que la différence* $-b + d$ *ne se perde pas.*

2. MATrix LABoratory, voir `www.themathworks.com`
3. Il s'agit du format classique de représentation de nombres à virgule flottante, avec environ 14 chiffres significatifs (Goldberg, 1991; Overton, 2001).

Malheureusement, quand des milliers d'opérations sont effectuées automatiquement par l'ordinateur, avec une même précision finie, ce contrôle manuel de l'erreur n'est plus possible. Pour l'exemple précédent, en fonction des valeurs numériques de a, b, c et de la précision utilisée, il est possible qu'une soustraction destructive (Higham, 1996) se produise et que la valeur obtenue de α soit très mauvaise. Voici un dernier exemple très illustratif extrait de (Moler *et al.*, 2003).

Exemple 4 *Supposons qu'on veuille calculer e^A, l'exponentielle de la matrice*

$$A = \begin{bmatrix} -49 & 24 \\ -64 & 31 \end{bmatrix}$$

en simple précision[4]*. L'algorithme classique consiste à développer la série de Taylor*

$$e^A = \sum_{j=0}^{\infty} \frac{1}{j!} A^j$$

jusqu'à que les termes soient inférieurs en valeur absolue à 10^{-7}. L'algorithme est facilement programmable et le résultat est

$$e^A \approx \begin{bmatrix} -22.2588 & -1.43277 \\ -61.4993 & -3.47428 \end{bmatrix}.$$

Malheureusement le résultat correct, arrondi à 6 chiffres significatifs, est

$$e^A \approx \begin{bmatrix} -0.735759 & 0.551819 \\ -1.47152 & 1.10364 \end{bmatrix}$$

et donc même les signes de certains éléments calculés par l'ordinateur sont incorrects. En observant chaque terme de la série on peut noter que quelques termes du milieu sont proches de 10^{-7} mais de signe contraire. Par conséquent, des soustractions destructives (Higham, 1996) *ont lieu et l'information significative est perdue.*

Des erreurs numériques sont commises, donc, en effectuant des opérations arithmétiques avec précision finie ou en appliquant des algorithmes non fiables ou de façon inadéquate sur un problème quelconque. Quand ces erreurs sont commises dans le monde réel, au moment d'analyser ou de contrôler un système physique, les conséquences peuvent être catastrophiques. Le 25 Février 1991 un missile Patriot n'a pas pu intercepter un SCUD Iraqien qui a détruit des bâtiments Américains en tuant 28 soldats. La cause : une mauvaise manipulation des erreurs d'arrondi (Skeel, 1992). Le 4 Juin 1996, le lanceur Ariane 5 a explosé après 45 secondes de vol avec un chargement évalué à 500 millions de dollars. La cause : un dépassement[5] *–overflow–* dans l'algorithme du système de référence d'inertie de la fusée (Gleick, 1996).

4. Environ 6 chiffres significatifs (Goldberg, 1991; Overton, 2001).
5. Un nombre sur 64 bits en virgule flottante supérieur à 65536 a été transformé en un entier sur 16 bits.

Heureusement, la plupart des erreurs numériques sont commises en bureau d'étude : pendant la simulation et la conception des systèmes. Ces erreurs n'ont comme conséquence qu'une perte de temps. Cependant, l'analyse et la compréhension de ces erreurs peuvent nous éviter des conséquences plus graves. Donc, il est nécessaire de disposer d'une théorie permettant d'analyser et de développer des algorithmes qui prennent en considération les problèmes numériques dus à la précision finie.

La communauté scientifique a accepté ce défi il y a longtemps, plusieurs disciplines scientifiques ont développé une branche d'analyse numérique (Hämmerlin *et al.*, 1999). Dans le cadre de cette travail, l'analyse numérique est considérée comme la théorie mathématique des méthodes qui peuvent être exécutées numériquement (dans un ordinateur par exemple).

2. Les mathématiques numériques et la commande

L'analyse de l'erreur arrière[6] (*backward error*) est un des éléments clés de l'analyse numérique. Le concept d'erreur arrière a été introduit pour la première fois par Von Neumann et Turing (Neumann *et al.*, 1947; Turing, 1948), mais sans doute l'étude la plus complète sur l'effet des erreurs d'arrondi est celle de Wilkinson (Wilkinson, 1965; Wilkinson, 1994). L'étude de la sensibilité[7] du problème analysé est aussi très importante pour déterminer la qualité de la solution obtenue par l'algorithme, autrement dit la magnitude de l'erreur avant (*forward error*). La première théorie du conditionnement[8] a été développée par Rice (Rice, 1966).

L'algèbre linéaire numérique est un exemple très illustratif de discipline née de l'intérêt et de la nécessité d'appliquer la théorie mathématique (l'algèbre linéaire) au monde des coefficients à précision finie où des résultats numériques sont requis. Cette discipline est constituée d'algorithmes numériques pour résoudre des problèmes dont la formulation est parfois relativement simple. Par exemple, le problème des valeurs propres (eigenvalues), dont la théorie est connue depuis longtemps, est encore actuellement un des problèmes numériques les plus délicats (Wilkinson, 1965).

Pendant le siècle dernier, l'algèbre linéaire numérique a été très développée. La production d'algorithmes numériques a été très vaste. En 2000 l'IEEE Computer Society et l'Institut Américain de Physique ont compilé une liste des 10 meilleurs algorithmes du siècle (Cipra, 2000; Dongarra *et al.*, 2000). Parmi ces 10 algorithmes, 4 sont des algorithmes numériques d'algèbre linéaire ou appliqués à l'algèbre linéaire :

– Les méthodes itératives sur les sous-espaces de Krylov (méthode de Lanczos, méthode du gradient, etc) en 1950.

6. Le reflet de toutes les erreurs d'arrondi commises par un algorithme numérique sur les données d'entrée (Higham, 1996; Wilkinson, 1965).
7. L'évolution du résultat d'un problème suite aux changements des données d'entrée (Higham, 1996).
8. Mesure de la sensibilité d'un problème (Higham, 1996; Higham *et al.*, 2004).

– L'approche de factorisation des matrices de Alston Householder en 1951.

– L'algorithme de compilation pour FORTRAN en 1957.

– L'algorithme QR de J.G.F. Francis en 1959.

Sans doute un des algorithmes les plus importants est l'algorithme de compilation de FORTRAN. Avec le langage de programmation FORTRAN[9] les scientifiques et ingénieurs ont pu finalement dire à l'ordinateur exactement ce qu'il doit faire sans passer des heures en programmant en langage machine. Le calcul scientifique est né sur la base de l'algèbre linéaire numérique et la programmation en FORTRAN (Press *et al.*, 1992).

Actuellement les bibliothèques comme LAPACK[10], successeur d'EISPACK et de LINPACK, contiennent de nombreuses routines programmées en FORTRAN pour résoudre des problèmes divers de l'algèbre linéaire comme par exemple : les valeurs propres, la factorisation SVD, QR, etc. (Anderson *et al.*, 1999). La bibliothèque LAPACK est basée sur des sous-programmes en FORTRAN pour effectuer des opérations basiques sur les vecteurs et les matrices. Cet ensemble de sous-programmes est appelé BLAS[11] (Lawson *et al.*, 1979).

La performance de LAPACK et BLAS est remarquable. Grâce à cette performance, ces bibliothèques d'algèbre linéaire numérique sont utilisées comme des "boîtes noires" à la base de logiciels de plus haut niveau de programmation comme par exemple Matlab ou Scilab[12] (Inria, 2003). Une autre caractéristique très importante de LAPACK et de BLAS est leur portabilité, c'est-à-dire leur indépendance de la plate-forme ou du système opérationnel utilisé. Cette portabilité a été favorisée par la création, en 1985, du standard ANSI/IEEE 754 pour les modèles arithmétiques en virgule flottante (Moler, 1996).

Il est bien connu qu'un phénomène qui a contribué énormément au développement de l'algèbre linéaire numérique sont les relations symbiotiques que cette discipline a eu avec d'autres disciplines en science et en ingénierie. Un exemple très clair est la théorie de la commande des systèmes linéaires représentés sous la forme matricielle (espace d'état). D'une part la théorie des systèmes est pleine de problèmes d'algèbre linéaire numérique, et d'autre part beaucoup d'algorithmes de l'algèbre linéaire numérique trouvent des applications intéressantes dans la théorie des systèmes. Cette interaction entre algèbre linéaire numérique et commande a été très fructueuse. Actuellement on trouve LAPACK et BLAS à la base de logiciels scientifiques pour la conception et la solution de problèmes en commande, constituant les logiciels de *CACSD (Computer Aided Control System Design)* (Herget *et al.*, 1982).[13]

---

9. Langage continuellement actualisé, voir `www.fortran.com`
10. Linear Algebra PACKage, voir `www.netlib.org/lapack`.
11. Basic Linear Algebra Sub-programmes, voir `www.netlib.org/blas`.
12. SCIentific LABoratory, voir `www.inria.fr/scilab`.
13. Dans le cadre d'IEEE, la Control System Society anime depuis 1982 un comité technique sur les logiciels de CACSD : IEEE CSS TC on CACSD, voir `www.laas.fr/cacsd`.

L'approche espace d'état (Kailath, 1980; Rosenbrock, 1970), suite aux travaux de Rudolf Kalman dans les années 1960, est devenue donc l'approche largement préférée en ce qui concerne l'obtention de résultats numériques. Beaucoup de logiciels de CACSD ont été développés dans ce cadre. On approfondit ce sujet dans le chapitre 3.

Par contre, d'autres disciplines mathématiques ont moins développé leurs côtés numériques. C'est le cas de l'algèbre non-linéaire et de la géométrie algébrique qui sont restés assez théoriques (Stetter, 2004). Pourtant, actuellement il est devenu de plus en plus nécessaire d'étendre cette théorie à des résultats numériques. Les polynômes sont devenus des outils naturels pour modéliser les systèmes dynamiques, qui, dans le monde réel, ont des coefficients numériques avec une précision limitée.

Grâce à la capacité des ordinateurs modernes -surtout au niveau de mémoire de stockage- le calcul symbolique (*symbolic computing*) est devenu possible et l'algèbre calculatoire (*computer algebra*) a été développée comme outil informatique pour faire des mathématiques pures avec l'ordinateur. Quelques exemples de logiciels qui permettent le calcul symbolique sont : Maple[14] (Waterloo-Maple, 2005) et Mathematica[15] (Wolfram-Research, 2005). Cependant, les résultats obtenus par le calcul symbolique sont en général inadéquats pour les problèmes réels. Par exemple, le résultat d'un système d'équations à coefficients numériques peut être donné comme des fractions d'entiers avec des centaines de chiffres. Le passage au monde numérique n'est pas toujours adéquat.

Des efforts pour raccourcir le chemin entre l'algèbre calculatoire et l'analyse numérique sont faits ces dernières années. Différents groupes de recherche dans les universités ont commencé à développer des logiciels numériques pour les problèmes sur les polynômes[16]. Quelques bibliothèques, comme par exemple NAG[17], ont été développées. Ces bibliothèques incluent plusieurs routines pour résoudre des problèmes numériques qui ne sont pas de l'algèbre linéaire, comme par exemple : l'extraction de racines de polynômes, les équations différentielles, l'interpolation, l'optimisation, etc. Des logiciels comme Maple basent leurs capacités numériques sur les bibliothèques NAG. Malheureusement, les bibliothèques NAG ne sont pas de libre distribution (contrairement à LAPACK ou BLAS) et il reste encore beaucoup de travail à faire, tant au niveau théorique que pratique, pour pouvoir parler d'une algèbre non-linéaire numérique.

Par rapport à la théorie de la commande des systèmes, ces dernières décennies d'autres approches ont été développées comme alternatives à l'approche espace d'état : l'approche des équations polynomiales (*polynomial equation approach*) (Kučera, 1979) et l'approche comportementale (*behavioural approach*) (Polderman *et al.*, 1998). Avec ces approches, qu'on regroupe ici sous le terme d'approche polynomiale, les systèmes sont représentés à l'aide de polynômes et de matrices polynomiales. Il devient alors nécessaire de réaliser plusieurs tâches numériques comme

14. MAthematical PLEasure, voir `www.maplesoft.com`.
15. Voir `www.wolfram.com`.
16. Voir par exemple `www.dm.unipi.it/~ananum/polynomial.html`. (Bini *et al.*, 1994)
17. Numerical Algorithms Group, voir `www.nag.co.uk`.

la manipulation des polynômes et des matrices polynomiales, la solution d'équations polynomiales et l'extraction de propriétés structurelles.

Malheureusement, et malgré le fait que l'approche polynomiale est actuellement un des plus puissants outils théoriques pour la commande des systèmes (voir par exemple (Hunt, 1993)), au niveau des logiciels de CACSD son développement est beaucoup plus faible que celui de l'approche espace d'état. La raison principale, comme on le verra dans les chapitres suivants, est le manque d'une théorie numérique consolidée pour les polynômes et les matrices polynomiales, et par conséquent un manque de logiciels de base comme LAPACK pour l'algèbre linéaire numérique.

Récemment, sous le pseudonyme d'algèbre polynomiale numérique (*numerical polynomial algebra*), la théorie de l'analyse numérique a finalement été adaptée aux polynômes par Hans Stetter (Stetter, 2004). Il s'agit d'une des premières contributions vers la consolidation d'une algèbre non-linéaire numérique. Dans la communauté de la commande de systèmes, beaucoup d'enthousiastes des méthodes polynomiales ont fait aussi des efforts pour combler la manque de logiciels de CACSD pour l'approche polynomiale (voir les chapitres suivants). Plusieurs algorithmes modernes pour les polynômes et les matrices polynomiales sont maintenant basés sur des opérations stables de l'algèbre linéaire numérique, voir (Basilio *et al.*, 2004; Henrion *et al.*, 1999; van Dooren *et al.*, 1983) par exemple. Des résultats récents comme (Stetter, 2000; Zeng, 2005), bien qu'ils ne soient pas encore très diffusés et appliqués, prédisent que beaucoup de problèmes polynomiaux mal conditionnés ou mal posés peuvent être résolus de manière satisfaisante dans le sens de l'analyse numérique et de l'algèbre polynomiale numérique. Cependant, il manque encore du travail pour qu'on puisse appliquer la théorie de l'analyse numérique aux problèmes polynomiaux en commande et pour que le développement de logiciels de CACSD pour l'approche polynomiale soit comparable à celui de l'approche espace d'état. On approfondit ce sujet dans le chapitre 4.

Les critères pour juger de la supériorité d'un logiciel de CACSD ne sont pas établis d'une façon unifiée et acceptée par tout le monde. Chaque concepteur ou utilisateur souligne certains critères plus que d'autres et base ses points forts sur différentes caractéristiques. Cependant, quelques réflexions générales peuvent se faire à ce propos :

– Les logiciels de CACSD peuvent être regroupés en deux grandes catégories. Le niveau inférieur contient les logiciels numériques comme les bibliothèques LAPACK, BLAS, etc. et les fonctions basiques programmées avec un langage générant du code machine (FORTRAN est le plus utilisé). Le niveau supérieur contient les procédures d'analyse et de synthèse de commande mises en ouvre en appelant des fonctions de plus bas niveau. Ce niveau supérieur constitue surtout l'interface qui permet à l'utilisateur d'appliquer les outils numériques pour résoudre son problème de commande.

– Aucun logiciel de CACSD ne peut être plus performant que le logiciel numérique sur lequel il est construit. Dans ce sens, la meilleure précision des opérations qu'on peut espérer provient des routines de niveau inférieur. Cependant, cette précision n'est pas toujours obtenue et cela dépend, entre autres, de la façon et du moment auquel le

logiciel numérique de base est utilisé : si la plupart du calcul numérique est mis en oeuvre à bas niveau, le temps d'exécution sera sûrement plus court et il y aura plus de chances pour qu'une précision satisfaisante soit obtenue.

– L'interface est un aspect qui peut différencier un logiciel CACSD d'un autre et qui est particulièrement importante pour l'utilisateur qui n'est pas familiarisé avec les problèmes et les algorithmes numériques. Certains logiciels de CACSD fournissent leur propre interface (Herget *et al.*, 1982), d'autres logiciels utilisent Matlab, par exemple, pour créer cette interface avec l'utilisateur. En tout cas, notons qu'une bonne interface rend plus facile l'utilisation mais peut aussi compromettre largement le temps d'exécution et parfois même la précision.

3. Approche espace d'état

Le processus de la commande d'un système dynamique commence par sa modélisation en termes d'équations mathématiques. En particulier, dans le cas des systèmes linéaires invariants dans le temps, on a un ensemble de p équations différentielles du type

$$a_{1d_1}y_1^{(d_1)}(t) + \cdots + a_{10}y_1(t) + \cdots + a_{pd_P}y_p^{(d_p)}(t) + \cdots + a_{p0}y_p(t) =$$

$$b_{1f_1}u_1^{(f_1)}(t) + \cdots + b_{10}u_1(t) + \cdots + b_{mf_m}u_m^{(f_m)}(t) + \cdots + b_{m0}u_m(t) + b_m \quad [1]$$

où $\alpha^{(q)}(t)$ représente la q-ième dérivée du signal scalaire $\alpha(t)$ par rapport au temps. Les paramètres a_{ij} et b_{kl} pour $i = 1:p$, $j = 0:d_i$, $k = 1:m$ et $l = 0:f_k$ sont constants. Par convention, toutes les variables sur lesquelles on a un contrôle et qui serviront à commander le système sont appelées les entrées $u_i(t)$, $i = 1:m$. D'autre part, toutes les variables sur lesquelles on observe l'effet de la commande sont appelées les sorties $y_i(t)$, $i = 1:p$.

Il existe différentes approches pour interagir avec ce modèle mathématique. Avec chaque approche, le modèle est représenté ou reformulé d'une façon particulière où les différentes propriétés du système sont plus ou moins dévoilées. On peut par exemple écrire chaque équation de type [1] comme un ensemble d'équations différentielles d'ordre 1 à l'aide de l'inclusion de variables d'état (Kailath, 1980). Après des manipulations algébriques, le modèle du système peut être écrit de la manière suivante :

$$x^{(1)}(t) = Ax(t) + Bu(t),$$
$$y(t) = Cx(t) + Du(t). \quad [2]$$

Le vecteur $x(t) \in \mathbb{R}^n$ est le vecteur des variables d'état et sert donc à relier le vecteur d'entrées $u(t) \in \mathbb{R}^m$ et le vecteur de sorties $y(t) \in \mathbb{R}^p$. Les matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$ et $D \in \mathbb{R}^{p\times m}$ sont constantes. La représentation [2] est utilisée par l'approche espace d'état (Kailath, 1980; Rosenbrock, 1970). Notons qu'en appliquant la transformée de Laplace sur la représentation [2] on peut éliminer $x(t)$ et en déduire la fonction de transfert $T(s)$ du système (Kailath, 1980)

$$y(s) = [C(sI - A)^{-1}B + D]u(s) = T(s)u(s)$$

dans l'hypothèse de conditions initiales nulles.

L'objet mathématique du modèle en space d'état sont des matrices constantes à valeurs réelles. Par conséquent, l'algèbre linéaire et l'algèbre linéaire numérique sont les outils mathématiques de base pour l'analyse et la synthèse de la commande des systèmes décrits par l'approche espace d'état. L'interaction entre algèbre linéaire et commande a été très fructueuse ces dernières années. Beaucoup de problèmes numériques en commande sont résolus de manière satisfaisante grâce à l'application des méthodes numériques de l'algèbre linéaire. L'utilisation des librairies numériques comme LAPACK ou BLAS a permis la conception de centaines de logiciels de CACSD. Plusieurs logiciels furent initialement présentés dans un numéro spécial de la revue IEEE Control Systems Magazine en 1982 (Herget *et al.*, 1982). Actuellement, certains de ces logiciels sont mis à jour et de nouveaux apparaissent. On peut mentionner, par exemple, la *Control System Toolbox* pour Matlab[18] et la bibliothèque *Slicot*[19] qui sont deux outils assez connus et utilisés.

L'utilisation de logiciels de CACSD a permis d'étendre l'utilisation de l'approche espace d'état dans la communauté. Ainsi, beaucoup de gens ont pu appliquer de complexes théories de commande en obtenant des résultats numériques mais sans s'occuper de la problématique numérique et algorithmique. Indiscutablement c'est très positif. Cependant, il faut rappeler que derrière tout logiciel il y a beaucoup de travail et de développement. D'une part il y a la théorie des mathématiques appliquées (l'algèbre linéaire numérique) qui n'est pas propre à l'approche espace d'état, et d'une autre part l'adaptation des résultats et algorithmes classiques en commande pour prendre en compte les problèmes numériques. Par la suite on illustre l'application des algorithmes classiques en commande, en utilisant les outils de l'algèbre linéaire numérique, pour faire face aux problèmes numériques et obtenir des résultats satisfaisants.

3.1. *Application des logiciels de CACSD*

Pour atteindre les objectifs de cette section, on étudie un des problèmes classiques en commande : le placement de pôles. Les algorithmes qu'on présente ici sont décrits à titre illustratif. Les résultats originaux peuvent se trouver, par exemple dans (Higham *et al.*, 2004; Kailath, 1980; Laub, 1985; Petkov *et al.*, 1991; van Dooren, 2004) et leurs références. Un autre objectif de cette section est de présenter de manière pratique les concepts d'analyse numérique qu'on utilisera dans le reste de ce travail. Pour un approfondissement sur ces concepts voir par exemple (Golub *et al.*, 1996; Higham, 1996; Stewart, 1998; Wilkinson, 1965; Wilkinson, 1994).

Étant donné un système sous la forme [2], l'objectif du placement de pôles est de trouver une commande linéaire statique $u = Fx$ telle que le système en boucle

18. Voir `www.mathworks.com/products/control`.
19. Créée par The Numerics in Control Network NICONET. Voir `www.win.tue.nl/niconet`.

fermée ait des pôles donnés, c'est-à-dire, telle que la matrice $A+BF$ ait le polynôme caractéristique désiré

$$\det(sI - A - BF) = s^n + f_{n-1}s^{n-1} + \cdots + f_0 = f(s).$$

3.1.1. *Algorithme classique et ses problèmes numériques*

Théoriquement ce problème n'est pas compliqué. Une manière méthodique de le résoudre consiste à exprimer la forme canonique

$$\bar{A} = T^{-1}AT = \begin{bmatrix} -a_{n-1} & -a_{n-1} & \cdots & -a_0 \\ 1 & 0 & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix}, \qquad \bar{B} = T^{-1}B = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad [3]$$

avec la transformation linéaire

$$T = \mathcal{CT} = \begin{bmatrix} B & AB & \cdots A^{n-1} \end{bmatrix} \begin{bmatrix} 1 & a_{n-1} & \cdots & a_1 \\ & 1 & \ddots & \vdots \\ & & \ddots & a_{n-1} \\ & & & 1 \end{bmatrix}, \quad [4]$$

et de choisir

$$\bar{F} = FT = \begin{bmatrix} a_{n-1} - f_{n-1} & \cdots & a_0 - f_0 \end{bmatrix}.$$

De cette façon, notons que

$$\det(sI - \bar{A} - \bar{B}\bar{F}) = f(s)$$

et, étant donné que T est une transformation de similarité,

$$\det(sI - A - BF) = f(s).$$

Ainsi, algorithmiquement, le problème du placement des pôles est réduit à la solution du système d'équations $FT = \bar{F}$ où T et $\bar{F}$ sont donnés et F est à déterminer, c'est la formule d'Ackermann (Ackermann, 1972). Donc les problèmes numériques que l'on peut rencontrer sont ceux d'un système d'équations du type $My = b$, où M dénote à présent la matrice T, et $b = \bar{F}$.

Soit $M \in \mathbb{R}^{n\times n}$ et $b \in \mathbb{R}^n$. Le problème qui consiste à déterminer $y = f(x) = f(M, b) \in \mathbb{R}^n$ est un problème mal posé au sens classique de Hadamard (Hadamard, 1902). En fait y ne dépend pas de manière continue des données x. Il existe des points de discontinuité ou singularités. Pour le cas carré, il s'agit de toutes les matrices M de rang plus petit que n telles que le système n'a pas de solution ou que la solution n'est pas unique. Donc, pour que le problème ait un sens, c'est-à-dire qu'il soit bien posé, on restreint l'espace des données d'entrée aux points réguliers uniquement. La

question qu'on se pose alors est comment mesurer la distance à la singularité d'un point x ? L'analyse de la sensibilité donne une réponse à ces questions.

La sensibilité de y par rapport aux perturbations sur x est mesurée par le conditionnement $\kappa(x)$. Intuitivement on peut conclure que si x est proche d'une singularité alors $\kappa(x)$ est grand car, dans ce cas, un petit changement dans x peut faire que y n'existe plus. Donc, $1/\kappa(x)$ est une mesure de la distance à la singularité. Si $\kappa(x)$ est grand on dit que le problème pour x donné est mal conditionné ou que le point x est mal conditionné. Par contre, le problème est bien conditionné si $\kappa(x)$ est petit.

La connaissance du conditionnement d'un problème est importante car elle peut nous donner un indice sur la qualité de la solution calculée. Une fois qu'un problème $y = f(x)$ et son conditionnement $\kappa(x)$ sont bien définis, on peut estimer la magnitude de l'erreur en la solution calculée $\hat{y}$, appelée erreur en avant (forward error) grâce à l'expression suivante :

$$\|y - \hat{y}\| \leq \kappa(x)\|e\| \qquad [5]$$

où e, l'erreur en arrière (backward error) est l'erreur introduite par l'algorithme appliqué sur les données d'entrée. Le conditionnement d'un même problème peut être différent pour différents types de perturbations. Alors, en fonction du type de perturbation introduit par l'algorithme, on devra utiliser la définition de conditionnement adéquate pour estimer la qualité de la solution obtenue.

Pour illustrer ces idées on retourne au problème de placement de pôles et la solution de $My = b$, en supposant M régulière, c'est-à-dire inversible. Le conditionnement de la matrice M pour le problème $My = b$ est donné par

$$\kappa_p(M) = \|M^{-1}\|_p \|M\|_p \qquad [6]$$

pour des perturbations arbitraires limitées en norme p, ou par

$$\kappa_*(M) = \||M^{-1}||M|\|_\infty \qquad [7]$$

pour le cas où chaque élément de M et/ou b est perturbé[20].

Exemple 5 *Pour résoudre un problème de placement de pôles, on veut calculer la solution y du système $My = b$ avec*

$$M = \begin{bmatrix} -100 & 0 & 0 \\ 0 & 85 & -52 \\ 0 & -0.003 & -0.003 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 85 \\ -0.003 \end{bmatrix}.$$

Supposons que les calculs soient effectués dans un système arithmétique à virgule flottante à 16 bits de précision, c'est-à-dire, avec une unité d'arrondi $\mu = 0.5\varepsilon \approx$

20. En général M^{-1} n'est pas connu ou il est coûteux de l'obtenir en termes calculatoires. Alors l'estimation du conditionnement par des méthodes approximatives est nécessaire (Higham, 1996). LAPACK contient de très bonnes méthodes pour estimer le conditionnement d'une matrice (Anderson *et al.*, 1999).

7.63×10^{-6}. *Le paramètre ε est la distance entre le nombre 1 et le nombre supérieur le plus proche. Dans cette travail, ε sera dénommé précision machine. Alors, pour des raisons pratiques, on considère les résultats donnés par Matlab (avec une unité d'arrondi $\mu_m = 0.5\,$`eps`$\approx 1.11 \times 10^{-16}$) comme exacts. Considérons qu'on applique un algorithme stable*[21] *(*backward stable*) c'est-à-dire que la solution calculée $\hat{y}$ est la solution exacte du système légèrement perturbé*

$$(M + \Delta)\hat{y} = b,$$

où Δ est une perturbation arbitraire (erreur arrière) telle que

$$\|\Delta\|_\infty \leq O(\mu)\|M\|_\infty.$$

Pour ce type de perturbation, la matrice M est très mal conditionnée avec $\kappa_\infty(M) = 28334.333...$ Alors, même si la méthode numérique utilisée pour résoudre le système $My = b$ est stable, étant donnée l'équation [5]*, il peut arriver que l'erreur avant $\|y - \hat{y}\|$ soit grande. Soit*

$$\Delta = \mathrm{rand}(3,3)\|M\|_\infty, \quad avec \quad \|\Delta\|_\infty = 0.0017... \leq 0.002... = \mu\|M\|_\infty$$

l'erreur arrière introduite par l'algorithme. Alors, $\hat{y}$ est la solution exacte

$$\hat{y} = (M + \Delta)^{-1}b = \begin{bmatrix} 1.499... \times 10^{-6} \\ 1.111... \\ 0.182... \end{bmatrix}.$$

La borne supérieure de l'erreur avant est donc $\kappa_\infty(M)\|\Delta\|_\infty = 49.213...$ ce qui n'incite pas à faire confiance à la qualité du résultat. Cependant, notons que cette borne est en fait assez pessimiste car le résultat n'est pas si mauvais que ça

$$\|y - \hat{y}\|_\infty = 0.182... \ll 49.213...$$

Maintenant supposons que

$$\bar{M} = PM = \begin{bmatrix} -100 & 0 & 0 \\ 0 & 85 & -52 \\ 0 & -72.733151 & -72.7331151 \end{bmatrix}, \quad \bar{b} = Pb = \begin{bmatrix} 0 \\ 85 \\ -72.7331151 \end{bmatrix}.$$

La matrice $\bar{M}$ est bien conditionnée, $\kappa_\infty(\bar{M}) = 2.302...$ Soit

$$\bar{\Delta} = \Delta\|\bar{M}\|_\infty\|M\|_\infty^{-1} \quad avec \quad \|\bar{\Delta}\|_\infty = 0.0018... \leq 0.0022... = \mu\|\bar{M}\|_\infty$$

l'erreur arrière introduite par l'algorithme. Alors, $\hat{y}$ est la solution exacte

$$\hat{\bar{y}} = (\bar{M} + \bar{\Delta})^{-1}\bar{b} = \begin{bmatrix} 1.212... \times 10^{-7} \\ 0.999... \\ 1.121... \times 10^{-5} \end{bmatrix}.$$

21. Plusieurs algorithmes stables pour la solution des systèmes d'équations sont présentés dans (Golub *et al.*, 1996) par exemple.

Dans ce cas, le borne supérieure de l'erreur avant, donnée par $\kappa_\infty(\bar{M})\|\bar{\Delta}\|_\infty = 0.004...$*, est beaucoup plus optimiste. En plus, notons que* $\hat{\hat{y}}$ *est aussi une solution de* $My = b$ *mais, par conséquent, plus exacte :*

$$\|y - \hat{\hat{y}}\|_\infty = 1.121... \times 10^{-5} \leq 0.004...$$

La matrice P *est dénommée matrice de* pré-conditionnement *ou de* changement d'échelle *(*scaling *en anglais).*

3.1.2. *Pré-conditionnement ou changement d'échelle*

Les techniques de changement d'échelle sont couramment utilisées en algèbre linéaire numérique. Actuellement, plusieurs fonctions de Matlab ou LAPACK appliquent ces techniques, même si cela passe inaperçu pour l'utilisateur standard. Par exemple : la fonction `eig` de Matlab calcule, sauf si l'on indique le contraire, les valeurs propres d'une matrice après avoir appliqué un changement d'échelle pour équilibrer les normes de ses colonnes et de ses lignes (Osborne, 1960; Parlett *et al.*, 1969). La fonction `ss` de la Control System Toolbox renvoie un système [2] sous représentation d'état où la matrice A est automatiquement équilibrée de façon à ce qu'elle soit mieux conditionnée pour le problème des valeurs propres. Dans l'exemple 5 le changement d'échelle a servi d'une part à trouver une meilleure borne pour l'erreur avant, mais aussi on peut vérifier que la solution obtenue après le changement d'échelle est plus exacte[22].

Une manière de définir le conditionnement de M pour le problème $My = b$, également très utile, consiste à choisir la norme Euclidienne dans la relation [6] :

$$\kappa_2(M) = \|M^{-1}\|_2\|M\|_2 = \frac{\sigma_{\max}(M)}{\sigma_{\min}(M)}$$

où $\sigma_{\max}(M)$, $\sigma_{\min}(M)$ sont la plus grande et la plus petite valeur singulière de M respectivement. Cette relation met en évidence que $1/\kappa_2(M)$ est bien une mesure de la distance à la singularité du problème $My = b$. Autrement dit, si $\kappa_2(M)$ est grand, M est peut être singulière. Pour une unité d'arrondi $\mu = 0.5\varepsilon$, la matrice M est singulière si $\kappa_2(M) > 1/n\varepsilon$. De manière équivalente, l'approximation numérique du rang[23] de M est égal au nombre de ses valeurs singulières supérieures à $n\varepsilon\sigma_{\max}$. Les valeurs singulières de M pour l'Exemple 5 sont $\sigma_{\max} = 100$, $\sigma = 99.644...$ et $\sigma_{\min} = 0.0041...$ Alors, le rang de cette matrice, pour la précision utilisée, est seulement deux car $\sigma_{\min} < 3\varepsilon\sigma_{\max} = .0045...$ Par contre, après le changement d'échelle, $\sigma_{\min}(PM) = 96.719... > 0.019... = 3\mu\sigma_{\max}(PM)$ ce qui rend la matrice $\bar{M}$ régulière.

Par rapport au problème de placement de pôles, la restriction que la matrice T donnée par [4] soit de rang plein est naturelle car elle implique tout simplement que le

22. Ils existe d'autres techniques pour améliorer la solution calculée d'un système d'équations. Il s'agit des techniques d'affinement itératif (iterative refinement) (Higham, 1996; Stewart, 1998).
23. Voir la documentation de la fonction `rank` de Matlab, et la section 3.1.4 de ce travail.

système est commandable. Cependant, tester la commandabilité du système en observant les valeurs singulières de la matrice T ou $\mathcal{C}$ n'est pas une méthode fiable. Étant donnée sa construction, on peut attendre que les normes des colonnes de la matrice de commandabilité $\mathcal{C}$ soient très déséquilibrées pour une matrice d'état arbitraire de dimension grande. Par conséquent, on peut attendre un grand écart entre $\sigma_{\max}(\mathcal{C})$ et $\sigma_{\min}(\mathcal{C})$, c'est-à-dire, que la matrice $\mathcal{C}$ soit singulière pour la précision donnée.

Exemple 6 *Soit un système sous la forme* [2] *avec* $n = 7$, $m = 1$ *et les éléments de* A,b *choisis aléatoirement uniformément entre* 0 *et* 1. *Soit*

$$\mathcal{C} = \begin{bmatrix} b & Ab & \cdots & A^6 b \end{bmatrix}$$

sa matrice de commandabilité. On considère toujours une unité d'arrondi $\mu = 0.5\varepsilon \approx 7.63 \times 10^{-6}$. *Les valeurs singulières de* $\mathcal{C}$ *sont :*

$$\sigma_1 = 2180.718..., \quad \sigma_2 = 0.886..., \quad \sigma_3 = 0.300..., \quad \cdots \quad \sigma_7 = 0.000...$$

Alors, pour la précision donnée, $\mathcal{C}$ *est singulière car* $\kappa_2(\mathcal{C}) = 2462878.095... > 1/7\varepsilon = 9362.285...$ *En fait, notons qu'à partir de* $i = 4$, $\sigma_i < 7\varepsilon\sigma_1$. *Donc l'approximation numérique du rang de* $\mathcal{C}$ *est 3.*

On a vu que le changement d'échelle peut servir à régulariser notre problème $FT = \bar{F}$ pour trouver une solution plus ou moins exacte, cf. l'Exemple 5. Cependant, il faut considérer les remarques suivantes :

– Dans l'Exemple 5 il est relativement facile de déterminer la matrice de changement d'échelle P. En fait, il suffit d'observer que la décomposition en valeurs singulières de M est donnée par $M = I\Sigma V$. Alors, si on veut modifier la magnitude de $\sigma_{\min}(M)$ de telle sorte que $\sigma_{\min}(M) \approx \sigma_{\max}(M)$, il suffit de multiplier la dernière ligne de M par une constante adéquate. Dans ce cas $P = \mathrm{diag}\{1, 1, \kappa_2(M)\}$. Malheureusement, en général il est beaucoup plus compliqué de déterminer la matrice P.

– En plus, il existe une limite pour les effets du changement d'échelle : on peut vérifier que, avec un changement d'échelle optimal, $\kappa_\infty(PM) = \kappa_*(M)$, où le conditionnement est défini en [7]. Noter que la valeur de $\kappa_*(M)$ est indépendante du changement d'échelle, c'est-à-dire, $\kappa_*(M) = \kappa_*(PM)$, cf. (Higham, 1996, Chapitre 7). D'après les propriétés des normes, on peut vérifier que si $\kappa_*(M) > 1/\varepsilon$, alors la matrice PM pour toute P diagonale sera toujours singulière pour la précision donnée. Dans l'exemple 6 $\kappa_*(\mathcal{C}) = 2168168.553... > 65536 = 1/\varepsilon$ donc on ne pourra pas régulariser le problème avec un changement d'échelle même si on arrive à trouver la matrice P optimale.

Donc notons que malgré les techniques de changement d'échelle et l'existence d'algorithmes stables pour la solution du système $FT = \bar{F}$, une solution satisfaisante du problème de placement de pôles n'est pas assurée. En fait il faut comprendre que si un algorithme numérique est composé de plusieurs sous-algorithmes ou sous-problèmes, et que si l'un de ces sous-problèmes est mal conditionné ou mal posé,

alors la fiabilité du résultat final est compromise. Pour notre exemple, nous avons deux sous-problèmes, d'abord la transformation d'état qui emmène à la construction de la matrice T, et ensuite la solution du système $FT = \bar{F}$. Pour construire une matrice de transformation d'espace d'état mieux conditionnée et ainsi avoir plus de chances d'obtenir un bon résultat, d'autres techniques d'algèbre linéaire numérique sont utilisées, en particulier les transformations orthogonales (Datta, 1985).

3.1.3. *Transformations orthogonales*

Une matrice $Q \in \mathbb{R}^{n\times n}$ est orthogonale si $Q^TQ = QQ^T = I$. Ainsi, toutes les valeurs singulières de Q sont égales à 1. La matrice Q est donc bien conditionnée :

$$\kappa_2(Q) = 1, \qquad \kappa_*(Q) < \kappa_\infty(Q) < n.$$

Une autre propriété importante des matrices orthogonales est qu'elles conservent par multiplication le conditionnement κ_2 d'une matrice quelconque :

$$\kappa_2(AQ) = \kappa_2(QA) = \kappa_2(A).$$

Supposons qu'on applique une transformation d'état orthogonale Q telle que

$$\bar{A} = QAQ^T = \begin{bmatrix} -a_{n-1} & -a_{n-1} & \cdots & -a_0 \\ \alpha_1 & \times & & \\ & \ddots & \ddots & \\ & & \alpha_{n-1} & \times \end{bmatrix}, \qquad \bar{B} = QB = \begin{bmatrix} b_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad [8]$$

où $\times$ représente un élément quelconque possiblement différent de zéro. Supposons aussi que $\bar{F}$ soit choisie telle que la boucle fermée ait le polynôme caractéristique désiré

$$\det(sI - \bar{A} - \bar{B}\bar{F}) = f(s).$$

Alors, la solution au problème de placement de pôles est simplement $F = \bar{F}Q$. Reste à déterminer $\bar{F}$. À partir de la définition des valeurs et vecteurs propres, on peut vérifier que

$$\bar{A} + \bar{B}\bar{F} = V\Lambda V^{-1}$$

avec

$$\Lambda = \mathrm{diag}\{\lambda_1, \ldots, \lambda_n\}, \qquad V = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}$$

où λ_i, $i = 1:n$ sont les valeurs propres de $\bar{A} + \bar{B}\bar{F}$ (racines de $f(s)$) et v_i, $i = 1:n$ sont les vecteurs propres correspondants. Notons qu'on peut construire les vecteurs v_i à partir des $n-1$ dernières lignes de $\bar{A} + \bar{B}\bar{F}$

$$\begin{bmatrix} \alpha_1 & \times - \lambda_i & & \\ & \ddots & \ddots & \\ & & \alpha_{n-1} & \times - \lambda_i \end{bmatrix} v_i = \Gamma_i v_i = 0, \quad [9]$$

et donc, extraire facilement $\bar{F}$ en divisant la première ligne de $\bar{A} - V\Lambda V^{-1}$ par b_1.

La solution de [9] peut être obtenue aussi grâce à des transformations orthogonales et ainsi assurer de bonnes propriétés numériques. En particulier on peut appliquer sur Γ_i une factorisation LQ, duale de la factorisation QR (Golub *et al.*, 1996) pour obtenir une base de son espace nul.

Le système transformé [8] est nommé la forme commandable de Hessenberg (à comparer avec la forme commandable [3]). La matrice orthogonale Q n'est qu'une composition de matrices de Householder (Golub *et al.*, 1996) appliquées pour annuler les éléments désirés dans A et B (Datta, 1985).

Cette méthode pour le placement de pôles est présentée dans (Miminis *et al.*, 1982; Miminis *et al.*, 1988). Les idées dans ces articles sont à la base de plusieurs autres algorithmes, en particulier de ceux présentés dans (Kautsky *et al.*, 1985) sur lesquels est basée la fonction `place` de la Control System Toolbox.

3.1.4. *Commandabilité du système*

La matrice Q en [8] est après tout une transformation de similarité pour l'état du système. Alors, les propriétés du système original (A, B) sont aussi celles du système transformé $(\bar{A}, \bar{B})$. En particulier, notons que

$$\bar{\mathcal{C}} = \left[\begin{array}{cccc} QB & QAQ^TQB & \cdots & (QAQ^T)^{n-1}QB \end{array} \right] = Q\mathcal{C}. \qquad [10]$$

La commandabilité du système reste invariante car le rang de $\bar{\mathcal{C}}$ est égal au rang de $\mathcal{C}$. La différence est que l'algorithme orthogonal pour le placement pôles ne dépend pas de l'inversibilité de $\mathcal{C}$. Une fois que $\bar{F}$ est déterminée, on peut toujours calculer F. Alors, où intervient la condition que le système doit être commandable ? La réponse se trouve, bien sûr, dans la partie où la plupart du calcul numérique est effectué, c'est-à-dire, dans la solution du système [9] pour les n valeurs propres désirées. Notons que si Γ_i n'a pas de rang plein par lignes, alors le calcul de son espace nul devient un problème mal posé car une petite perturbation des éléments de Γ_i peut changer la dimension de l'espace nul. Alors, on requiert que Γ_i ait rang plein, autrement dit, que α_i, $i = 1 : n-1$ soient différents de zéro. Maintenant, si on observe soigneusement la matrice de commandabilité

$$\bar{\mathcal{C}} = \begin{bmatrix} b_1 & \times & \times & \times & \cdots & \times \\ & b_1\alpha_1 & \times & & & \times \\ & & b_1\alpha_1\alpha_2 & \times & & \times \\ & & & b_1\alpha_1\alpha_2\alpha_3 & & \vdots \\ & & & & \ddots & \times \\ & & & & & b_1\alpha_1\cdots\alpha_{n-1} \end{bmatrix} \qquad [11]$$

on peut vérifier que la condition $\alpha_i \neq 0$, $i = 1 : n-1$ implique que le système est commandable, $\mathrm{rank}(\mathcal{C}) = n$.

Finalement, à partir de [10] et [11] notons que $Q^T\bar{\mathcal{C}} = \mathcal{C}$. Alors, Q peut être obtenue à partir de la factorisation QR de $\mathcal{C}$. Pour des raisons numériques, on construit

Q directement à partir de A et B. Cependant, il est important de noter que la factorisation QR peut nous donner une approximation plus utile du rang de $\mathcal{C}$ (ou de la commandabilité du système) que celle obtenue à partir des valeurs singulières.

Exemple 7 *Considérons la matrice de commandabilité $\mathcal{C}$ de l'Exemple* 6. *En obtenant sa factorisation QR, on observe les valeurs suivantes sur la diagonale du facteur triangulaire $\bar{\mathcal{C}}$:*

-1.476..., 3.417..., -1.836..., 1.447..., -0.311..., -0.142..., 0.008...

Ainsi le système est commandable et on peut appliquer l'algorithme décrit ci-dessus pour le placement de ses pôles. Le système est commandable, mais ça n'empêche pas que $\mathcal{C}$ soit très mal conditionnée. Autrement dit, si par contre on veut placer les pôles avec la formule d'Ackermann, on n'obtiendra pas de résultats satisfaisants.

4. Approche polynomiale

Une autre manière de réécrire l'ensemble des p équations de type [1] qui représentent un système dynamique fait appel aux matrices polynomiales. En regroupant toutes les entrées et sorties dans les vecteurs correspondants u et y on obtient :

$$D(s)y(s) = N(s)u(s). \qquad [12]$$

Les matrices $D(s)$ et $N(s)$ sont des matrices polynomiales en la variable s, de dimensions $p \times p$ et $p \times m$ respectivement. Rappelons que dans le cas continu, la variable s est, traditionnellement, l'opérateur de Laplace. Si le système est à temps discret, alors la variable devient l'opérateur de décalage.

Étant donné que la fonction de transfert du système est unique (Kailath, 1980), on vérifie que

$$T(s) = D^{-1}(s)N(s) = C(sI - A)^{-1}B + D.$$

La représentation [12] est utilisée par l'approche polynomiale (Kučera, 1979; Polderman *et al.*, 1998).

Chaque représentation a ses points forts et ses points faibles, et implique une façon différente d'analyser le même système. Par conséquent chaque approche à la commande des systèmes a aussi ses particularités et utilise ses propres méthodes et outils mathématiques. Une discussion des avantages ou désavantages de chaque approche impliquerait beaucoup d'efforts pour n'en tirer aucune conclusion. En fait, le vrai avantage consiste à disposer de plusieurs façons différentes de résoudre un problème. Pour certains problèmes l'approche espace d'état sera plus efficace, pour d'autres problèmes l'approche polynomiale sera plus adéquate. Actuellement, au niveau théorique, quelle approche utiliser reste simplement une question de préférence personnelle. Cependant, au niveau des logiciels de CACSD, l'approche espace d'état est beaucoup plus développée.

Comme brièvement rappelé dans le chapitre antérieur, le succès des logiciels de CACSD par approche espace d'état est fortement basé sur la théorie bien consolidée de l'algèbre linéaire numérique et les logiciels de calcul numérique existants. Cependant, notons que cette théorie n'est pas propre à l'approche espace d'état. C'est le fait que le système soit représenté par des matrices et vecteurs constants qui nous permet de l'appliquer. Par contraste, avec l'approche polynomiale on représente le système sous la forme [12]. Les objets mathématiques du modèle sont alors les polynômes et les matrices polynomiales, et malheureusement il n'existe pas encore de théorie d'analyse numérique solide pour ces objets.

Le développement de logiciels de CACSD pour les polynômes est très restreint. À notre connaissance, parmi les logiciels existants qui permettent la manipulation des polynômes et matrices polynomiales[24] seulement la Polynomial Toolbox pour Matlab[25] (PolyX, 2000) est orientée vers les systèmes en commande et n'exécute quasiment que des calculs numériques, et non symboliques.

Quelques autres exemples de logiciels numériques de CACSD pour l'approche polynomiale sont issus des efforts de quelques groupes de chercheurs et étudiants[26] mais leur diffusion est encore très limitée et il manque une analyse rigoureuse des propriétés numériques. Comme pour l'approche espace d'état, mais à une petite échelle, la variété des interfaces, des méthodes et des types d'implémentation rend très difficile la définition d'un logiciel standard pour l'approche polynomiale. Par rapport à l'interface pour l'utilisateur, la conception d'un logiciel de CACSD pour les méthodes polynomiales implique de mettre en place une structure qui permet à l'ordinateur d'accepter, de stocker et de représenter les variables (indéterminées) des polynômes[27]. Il est clair que cette interface peut nécessiter des temps de calcul importants dans certains cas.

Exemple 8 *Prenons les cas de la multiplication* $C(s) = A(s)B(s)$ *avec* $A(s) = A_0 + A_1 s + \cdots \in \mathbb{R}^{15\times 15}[s]$ *et* $B(s) = B_0 + B_1 s + \cdots \in \mathbb{R}^{15\times 15}[s]$ *deux matrices polynomiales de degrés* 10 *et* 12 *respectivement. En général* $C(s) = C_0 + C_1 s + \cdots \in \mathbb{R}^{n\times n}[s]$ *a degré 22 et peut s'obtenir numériquement avec l'opération équivalente*

$$\begin{bmatrix} C_0 & \cdots & C_{22} \end{bmatrix} = \begin{bmatrix} A_0 & \cdots & A_{10} \end{bmatrix} \begin{bmatrix} B_0 & \cdots & B_{12} & & \\ & \ddots & & \ddots & \\ & & B_0 & \cdots & B_{12} \end{bmatrix}$$

sur les matrices réelles. Avec une machine SUN Microsystems Ultra 5 (SunOS 5.9) et Matlab 7, la version 3 de la Polynomial Toolbox a besoin de 16.3 secondes pour exécuter l'instruction `mtimes(A,B)` *et dévoiler le résultat, alors que le temps d'exécution de la multiplication matricielle réelle ci-dessus est seulement de 0.3 secondes.*

24. Par exemple : Maple, Mathematica, MACSYMA, Reduce, MuMath, etc...
25. Voir `www.polyx.com`.
26. Voir `http ://ar.c-a-k.cz/polynomial/software.html`.
27. Noter cependant qu'aucun calcul symbolique n'est considéré, on se restreint aux logiciels numériques.

Les 16 secondes de différence sont utilisées uniquement pour l'interface (majoritairement pour représenter $C(s)$ sous la forme d'une matrice avec ses éléments qui sont des polynômes).

Évidemment, une programmation soignée peut améliorer l'interface. On peut même la réduire au minimum (si l'utilisateur est initié) et présenter les matrices polynomiales comme un ensemble de coefficients.

La plupart des logiciels de CACSD pour l'approche polynomiale utilisent ou peuvent utiliser les bibliothèques numériques de base comme LAPACK ou BLAS. Malheureusement la capacité de ces bibliothèques numériques n'est pas toujours exploitée au maximum. Pour illustrer ceci prenons le cas de la multiplication de l'Exemple 8. LAPACK (BLAS) est appelé à travers Matlab quand la Polynomial Toolbox exécute l'instruction `mtimes(a,b)`. L'opération exécutée par l'ordinateur est une opération du niveau 3 de BLAS[28]. Notons, néanmoins, qu'en utilisant la capacité de LAPACK (BLAS) pour manipuler les matrices avec un motif de zéros défini (matrices creuses, *sparse matrices*) et avec une structure particulière (matrices structurées), le calcul de $C(s)$ pourrait être plus efficace. Ainsi, une meilleure exploitation des bibliothèques peut être assurée par l'intermédiaire d'une meilleure mis en ouvre (programmation).

4.1. *Problèmes numériques et polynômes*

Quel est le vrai problème des logiciels de CACSD pour l'approche polynomiale ? On essaiera de répondre à cette question dans les paragraphes suivants. Dans la littérature technique sur les problèmes numériques en commande, deux critiques sont généralement faites aux méthodes polynomiales :

1) Avec l'approche polynomiale, les problèmes d'analyse et de synthèse en commande des systèmes linéaires sont généralement réduits à l'analyse des propriétés des matrices polynomiales en tant qu'objets algébriques (obtention de la structure propre, le rang, etc.), la manipulation de ces matrices polynomiales (factorisation, triangularisation, etc.) et la solution d'équations polynomiales ou équations Diophantiennes (Kučera, 1993). Traditionnellement les méthodes pour accomplir ces tâches sont basées sur les opérations élémentaires dans l'anneau des polynômes et il est prouvé que ces opérations peuvent causer une instabilité numérique (Gantmacher, 1959; van Dooren, 2004).

2) Bien qu'au niveau théorique les représentations [2] et [12] sont équivalentes, au niveau numérique la réputation des polynômes comme objets de modélisation est plutôt mauvaise. Les polynômes sont vus, en général, comme des objets qui génèrent des problèmes mal conditionnés (voir mal posés) (Higham *et al.*, 2004).

28. Une opération du niveau 3 signifie une opération matrice–matrice.

L'instabilité est spécifique à l'algorithme utilisé. Dans ce sens là, l'instabilité est due à l'application des opérations élémentaires et non pas à l'approche polynomiale elle-même. On peut, donc, en utilisant d'autres types d'opérations, concevoir des algorithmes numériques stables. Malheureusement, la stabilité d'un algorithme n'assure pas l'exactitude de la solution calculée, cette propriété dépend aussi du conditionnement du problème, cf. Exemple 5. Le conditionnement des problèmes polynomiaux qu'on rencontre en commande (via l'approche polynomiale) est en général élevé. Cependant, comme on le verra par la suite, de nouveaux résultats laissent supposer qu'on peut contourner ces inconvénients.

L'instabilité potentielle des opérations élémentaires sur l'anneau des polynômes est concentrée dans le choix d'un *pivot*[29]. Le choix du pivot polynomial se base essentiellement sur son degré, et ne pas considérer ses coefficients peut introduire des erreurs numériques importantes.

Exemple 9 *On veut obtenir la forme de Smith $S(s)$ de la matrice polynomiale*

$$F(s) = \begin{bmatrix} s & 0 & 1 & \alpha \\ 0 & s & 1 & 1 \\ 1 & 0 & s+1 & 0 \\ 0 & 1 & 1 & s+1 \end{bmatrix}.$$

Après quelques opérations polynomiales élémentaires par lignes et par colonnes groupées dans $L_1(s)$ et $C_1(s)$ respectivement on obtient

$$L_1(s)F(s)C_1(s) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & -s & 0 \\ 0 & 1 & 0 & -s \end{bmatrix} F(s) \begin{bmatrix} 1 & 0 & -s-1 & 0 \\ 0 & 1 & -1 & -s-1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -s^2-s+1 & \alpha \\ 0 & 0 & 1-s & -s^2-2+1 \end{bmatrix} = S_1(s).$$

Jusqu'ici nous n'avons pas de problèmes numériques. Pour continuer jusqu'à l'obtention de la forme de Smith, on choisit comme pivot suivant l'élément de plus petit degré dans la sous-matrice $S_1(3:4, 3:4)$, c'est-à-dire α. Avec ce pivot on introduit des zéros dans les positions correspondantes. Ainsi on obtient

$$L(s)F(s)C(s) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & 0 \\ 0 & 0 & 0 & -\gamma s^4 - 2\gamma s^3 + \gamma s^2 + (2\gamma - 1)s + (1-\gamma) \end{bmatrix} = S(s)$$

29. Un élément sur une ligne ou colonne à partir duquel on peut introduire des zéros sur une matrice ou vecteur à l'aide d'opérations par lignes ou colonnes.

avec $\gamma = 1/\alpha$ *et*

$$L(s) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & -s & 0 \\ \gamma s^2 + \gamma s - \gamma & 1 & -\gamma s^3 - \gamma s^2 + \gamma s & -s \end{bmatrix},$$

$$C(s) = \begin{bmatrix} 1 & 0 & 0 & -s-1 \\ 0 & 1 & -s-1 & -\gamma s^3 - 2\gamma s^2 + \gamma - 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & \gamma s^2 + \gamma s - \gamma \end{bmatrix}.$$

Si α *est très petit, alors* γ *devient très grand. Dans ce cas notons que, suivant la précision utilisée, après l'arrondi inévitable de l'ordinateur, les matrices calculées deviennent* $\hat{L}(s) = L(s)$, $\hat{C}(s) = C(s)$ *et*

$$\hat{S}(s) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & 0 \\ 0 & 0 & 0 & -\gamma s^4 - 2\gamma s^3 + \gamma s^2 + 2\gamma s - \gamma \end{bmatrix}$$

ce qui démontre l'instabilité de l'algorithme. En fait la matrice $\hat{S}(s)$ *n'est pas la forme de Smith de* $A(s)$ *légèrement perturbée,*

$$\hat{L}^{-1}(s)\hat{S}(s)\hat{C}^{-1}(s) = \begin{bmatrix} s & 0 & 1 & \alpha \\ 0 & s & s & 1 \\ 1 & 0 & s+1 & 0 \\ 0 & 1 & 1 & s+1 \end{bmatrix} \neq F(s),$$

car l'information sur l'élément $(2,3)$ *de* $F(s)$ *s'est complètement perdue.*

Les opérations polynomiales élémentaires ont leur intérêt et leur importance car elles constituent l'outil mathématique avec lequel s'explique la théorie de l'approche polynomiale. Cependant, il est clair que malgré cet intérêt académique, les algorithmes polynomiaux classiques doivent s'adapter pour prendre en compte les problèmes numériques. Ce besoin d'adaptation a aussi été observé dans l'approche espace d'état et l'algèbre linéaire numérique, cf. la section 3.1. Notons que le phénomène illustré dans l'exemple antérieur ressemble à ce que l'on observe avec l'élimination Gaussienne (Golub *et al.*, 1996), où choisir un pivot uniquement par sa position sur la diagonale peut causer les mêmes problèmes numériques[30].

La solution donnée à ces problèmes par l'algèbre linéaire numérique se dénomme stratégie de pivot (*pivoting*) (Golub *et al.*, 1996). Il y a plusieurs stratégies de pivot

30. Considérer par exemple la matrice $\begin{bmatrix} \alpha & 1 \\ 1 & 1 \end{bmatrix}$ avec α très petit.

mais l'idée générale consiste à choisir comme pivot l'élément de plus grande magnitude. Malheureusement ces stratégies de pivot ne sont pas toujours applicables dans le cas des opérations polynomiales élémentaires, comme pour l'Exemple 9.

Pour contourner les problèmes de stabilité on doit alors éviter les opérations élémentaires en ne manipulant que les coefficients de la matrice polynomiale analysée, à l'aide des méthodes fiables de l'algèbre linéaire numérique. Pour l'Exemple 9 la solution est relativement facile. Notons tout d'abord que $A(s)$ est un faisceau[31] et donc que ses zéros finis peuvent être obtenus par la factorisation QZ, qui est une méthode backward stable (Golub *et al.*, 1996; Moler *et al.*, 1973). Dans notre exemple cela équivaut à trouver les valeurs propres de A_0. Ensuite, à l'aide de la forme canonique de Kronecker (van Dooren, 1979), on détermine la multiplicité algébrique et géométrique de chaque zéro, et avec cette information on construit les polynômes invariants de $A(s)$, c'est-à-dire les éléments sur la diagonale de sa forme de Smith.

Les mauvaises propriétés numériques des polynômes, en tant qu'objets de modélisation, sont liées principalement à la surcondensation des données. L'exemple suivant extrait de (Higham *et al.*, 2004) est très illustratif.

Exemple 10 *On veut calculer les valeurs propres de la matrice*

$$A = Q^T \mathrm{diag}\{1, 2, \ldots, 20\} Q$$

avec Q une matrice orthogonale quelconque. La matrice A est symétrique et donc par la factorisation QR symétrique (qui est stable (Golub et al.*, 1996)) on peut obtenir les valeurs propres de A avec une très bonne précision. Notons aussi que A est parfaitement bien conditionnée pour le problème des valeurs propres car ses valeurs propres coïncident avec ses valeurs singulières (Parlett* et al.*, 1969). Par contre, si on essaye de calculer les valeurs propres de A à partir de la définition classique, c'est-à-dire comme les racines du polynôme caractéristique $p(s) = \det(sI - A)$, les résultats obtenus seront très inexacts. Par exemple, avec la fonction* `roots` *de Matlab, les 3 plus grandes racines de $p(s)$ calculées sont : 20.0003, 18.9970 et 18.0117. Notons que les coefficients du polynôme caractéristique varient entre 1 et 2.43×10^{18} et donc on aura des problèmes pour les représenter avec une précision finie.*

La surcondensation des données est évidente : au lieu de considérer les 210 coefficients de la matrice A, on ne considère que 20 coefficients du polynôme $p(s)$. Malheureusement ces coefficients varient dans une amplitude que l'ordinateur ne peut pas manipuler avec précision. D'autre part, il faut noter que ce problème n'est pas propre aux polynômes, dans l'Exemple 6 on a vu comme les normes des colonnes d'une matrice peuvent varier aussi avec une amplitude très vaste.

Il est certain que la formulation du problème en termes polynomiaux n'est pas la meilleure méthode pour cet exemple. Cependant, se baser sur des exemples comme

31. Matrice polynomiale de degré un.

celui-ci pour en conclure qu'il vaut mieux éviter les polynômes, nous prive d'une alternative qui peut être tout aussi efficace qu'une autre, voire parfois meilleure. On peut toujours trouver des exemples pour lesquels les méthodes polynomiales ont une meilleure performance. Évidemment, pour chaque exemple il existera sûrement un contre-exemple, raison pour laquelle nous ne poursuivrons pas cette discussion que nous croyons sans intérêt.

Il est vraiment important de noter que, tout comme au niveau théorique chaque approche a ses avantages et désavantages et surtout sa propre procédure d'analyse, au niveau numérique on doit résoudre des problèmes différents qui doivent s'analyser de façon différente.

Reprenons l'Exemple 10, pour lequel on a décrit deux problèmes différents dont les solutions, avec une précision infinie, sont équivalentes. D'un côté on a le problème de l'obtention des valeurs propres pour lequel la matrice A est très bien conditionnée, et de l'autre côté le problème du calcul des racines du polynôme caractéristique. C'est en analysant ce dernier problème qu'on conclut que le polynôme $p(s)$ est mal conditionné pour le problème de l'obtention de ses racines. Par contre, si on compare les racines de $p(s)$ avec les valeurs propres de A, la seule chose qu'on peut conclure c'est que la méthode qui consiste à obtenir les valeurs propres d'une matrice en calculant les racines du polynôme caractéristique n'est pas stable.

En conclusion si on a décidé d'appliquer l'approche polynomiale ou s'il on a un problème représenté avec des polynômes, l'analyse doit être faite dans le cadre des polynômes. Cela dit, il est important de consolider une théorie d'algèbre polynomiale numérique qui adapte l'analyse numérique aux polynômes. On a vu que cette théorie est actuellement en train de se développer sur les bases de la géométrie algébrique et de l'algèbre commutative (Stetter, 2004).

4.2. *Méthodes pour les matrices polynomiales*

Il existe différentes approches pour résoudre numériquement les problèmes en commande par approche polynomiale. Ici nous les regroupons en deux grandes catégories[32] :

– Exprimer le problème polynomial comme un problème de commande d'un système en espace d'état et appliquer les méthodes numériques et logiciels de CACSD propres à cette approche, cf. la section 3.1. Ensuite la solution au problème en espace état est reformulée en termes du problème polynomial original. Un exemple clair de cette procédure est représenté par les algorithmes pour la factorisation spectrale des matrices polynomiales via la solution des équations de Riccati

32. Voir (Henrion, 1998) pour une classification plus détaillée. Rappelons qu'ici on parle uniquement des problèmes polynomiaux qui sont numériquement réductibles. Cependant ils existent des problèmes, qui n'ont pas cette propriété, pour lesquels le calcul symbolique doit être appliqué (Munro, 1999).

(Stefanovski, 2003; Abou-Kandil *et al.*, 2003). L'obtention du descripteur du système est aussi une procédure qui permet, via l'espace d'état, de résoudre des problèmes avec les matrices polynomiales (Varga, 2000; Varga, 2004). Voir aussi (Kraffer, 1996; Kraffer, 1998) pour d'autres algorithmes d'espace d'état pour les matrices polynomiales.

– Une façon plus directe de travailler consiste en manipuler directement les coefficients des matrices polynomiales pour résoudre un problème constant équivalent mais sans une reformulation en espace état. La linéarisation d'une matrice polynomiale via la construction de sa forme compagne est une procédure très utilisée. Noter que la forme compagne résultante de la linéarisation est un faisceau sur lequel on peut appliquer des algorithmes de l'algèbre linéaire numérique (van Dooren *et al.*, 1983), voir également le point 4.3 du rapport prospectif (Demmel *et al.*, 2005), ou encore (Ipsen, 2004). Cependant, strictement parlant, la linéarisation est aussi une réalisation d'état de la matrice polynomiale linéarisée. La construction de matrices de Sylvester[33] avec les coefficients des matrices polynomiales (Stefanidis *et al.*, 1992), par contre, est une procédure encore plus directe car le problème polynomial original est directement formulé via ses coefficients. L'interpolation et l'utilisation de la transformée rapide de Fourier sont aussi des méthodes prometteuses qui ne passent pas par l'espace état (Hromčík, 2004).

La structure générale de ces algorithmes est la suivante :

Algorithme 1 *(Algorithme général)*

1) Déterminer un problème constant équivalent (CEP) au problème polynomial original (OPP) sur lequel on peut appliquer des méthodes numériques ;

2) Résoudre le CEP avec des algorithmes fiables de l'algèbre linéaire numérique ;

3) Obtenir la solution du OPP à partir de la solution calculée du CEP.

Exemple 11 *Considérons le problème de l'obtention des zéros finis*[34] *de la matrice polynomiale*

$$A(s) = A_2 s^2 + A_1 s + A_0 = \begin{bmatrix} -1 - s + s^2 & -\alpha \\ -1 - s & -1 - s + s^2 \end{bmatrix}.$$

On peut démontrer que les valeurs propres généralisées du faisceau sous la forme compagne

$$F(s) = sF_1 + F_0 = s \begin{bmatrix} I & 0 \\ 0 & A_2 \end{bmatrix} + \begin{bmatrix} 0 & -A_0 \\ I & -A_1 \end{bmatrix}$$

sont égales aux zéros recherchés. Le premier pas de l'Algorithme 1 est la construction de $F(s)$. Ensuite les pas 2 et 3 consistent à obtenir la factorisation QZ de $F(s)$

$$\bar{F}_1 = QF_1Z, \qquad \bar{F}_0 = QF_0Z$$

33. Matrices structurées (Toeplitz ou Toeplitz par blocs) et constantes.
34. Les zéros finis ou valeurs propres de $A(s)$ sont les valeurs de s pour lesquelles $A(s)$ perd son rang.

et à évaluer le quotient des valeurs diagonales de $\bar{F}_0$ et $\bar{F}_1$ (Moler et al.*, 1973). Notons que pour cet exemple, la forme compagne de $A(s)$ résulte en la matrice $F(s)$ de l'Exemple 9. Donc, la solution à notre OPP est simplement l'ensemble des valeurs propres de F_0.*

La complexité[35] de l'Algorithme 1 dépend du nombre d'opérations effectuées à chaque pas. Par conséquent, on souhaite que le CEP admette une formulation simple et également que sa solution soit facilement traduite en la solution du OPP. L'idée est donc de concentrer le gros du calcul sur la solution du CEP. Cet objectif n'est pas toujours atteint, dans (Beelen *et al.*, 1987) par exemple, une méthode pour obtenir le space nul d'une matrice polynomiale est présentée, avec cette méthode, le calcul correspondant aux pas 1 et 3 de l'Algorithme 1 peut être plus coûteux que celui correspondante au pas 2, voir aussi (Zúñiga *et al.*, 2004a).

Pour analyser la stabilité de l'Algorithme 1 on le divise en 3 sous-algorithmes. Il est facile de montrer que si l'un des sous-algorithmes est instable, alors l'algorithme global est instable (Higham, 1996; Higham *et al.*, 2004). On s'intéresse donc à la stabilité de chaque sous-algorithme. En particulier, il est désirable que l'algorithme de solution du CEP (pas 2) soit stable. Malheureusement cet objectif ne garantit pas la stabilité de l'algorithme global. Comme rappelé dans la section 4.1., l'analyse doit être faite à partir des polynômes. Par conséquent, non seulement la solution du CEP, mais aussi l'erreur arrière de la méthode numérique appliquée au CEP doivent être traduites en termes des coefficients des polynômes ou des matrices polynomiales analysées. Parfois on verra que de petites erreurs dans les coefficients du CEP sont traduites en des erreurs considérables dans les coefficients du OPP (Tisseur *et al.*, 2001).

Il est possible d'obtenir une borne supérieure de l'erreur arrière pour le OPP à partir de l'erreur arrière pour le CEP (Zúñiga *et al.*, 2005). Cependant, on peut vérifier que cette borne supérieure est parfois pessimiste. C'est l'analyse de la géométrie du problème et l'algèbre polynomiale numérique qui permettront d'obtenir des bornes plus précises, ainsi que de comprendre la sensibilité (conditionnement) des polynômes ou des matrices polynomiales analysées.

Actuellement, l'étude du problème des valeurs propres d'une matrice polynomiale commence à appliquer ce type d'analyse. Dans (Edelman *et al.*, 1995) les auteurs montrent, via un approche géométrique, comme les perturbations sur les coefficients F_1 et F_0 du faisceau associé à la matrice polynomiale $A(s)$, cf. exemple 11, correspondent à des perturbations $\Delta(s)$ de premier ordre sur les coefficients de $A(s)$. Des expressions exactes pour $\Delta(s)$ sont établies. Quelques stratégies de pré-conditionnement basées sur ces résultats sont développées pour réduire la magnitude de $\Delta(s)$, voir par exemple (Lemonnier *et al.*, 2003; Lemonnier *et al.*, 2004). Il semble que le préconditionnement sur le faisceau $F(s)$ corresponds à un changement de la base utilisée

35. Estimation du nombre d'opérations élémentaires en virgule flottante (flops) effectuées par l'ordinateur en exécutant un algorithme. Cette estimation est représentée comme $O(n^N)$: de l'ordre de n^N où n est un paramètre du problème, comme la dimension, et N est un entier positif (Golub *et al.*, 1996).

pour représenter la matrice polynomiale $A(s)$. Cette technique est illustrée pour le problème quadratique dans (Fan *et al.*, 2004) par exemple. Dans (Zhang, 2001) il est montré qu'un changement de base peut améliorer le conditionnement d'un polynôme scalaire. La sensibilité du problème de valeurs propres d'une matrice polynomiale est étudiée dans (Dedieu *et al.*, 2003; Tisseur, 2000). Diffèrent expressions pour le conditionnement sont présentées. La relation parmi ce conditionnement et celui du faisceau $F(s)$ associé a été étudiée dans (Higham *et al.*, 2005) et (Mackey *et al.*, 2005). Il est prouvé comme quelques linéarisations ne sont pas adéquates car le faisceau résultante est plus mal conditionné que la matrice originale. Des techniques pour construire une linéarisation optimale sont aussi présentées. Toute cette théorie est malheureusement limitée aux problèmes bien posés et, comme signalé dans (Dedieu *et al.*, 2003), les matrices polynomiales avec des racines multiples sont infinitivement mal conditionnées. Quand des racines multiples sont considérées, l'analyse du problème directement sur les polynômes (avec l'algèbre polynomiale numérique (Stetter, 2004)) permet d'obtenir résultats satisfaisantes. Dans (Zeng, 2005) par exemple, une méthode numérique fiable pour calculer les racines multiples d'un polynôme scalaire est présentée.

On a vu que la linéarisation d'une matrice polynomiale est une procédure qui entre dans le cadre du pas 1 de l'Algorithme 1, cf. Exemple 11. Le CEP qu'on analyse ensuite se pose sur le faisceau obtenu et peut être résolu en appliquant des méthodes numériques comme l'obtention d'une forme canonique de Kronecker, ou une factorisation QZ par exemple (van Dooren, 1979). Cette approche appelée par la suite l'approche des faisceaux est très répandue dans la littérature technique, en particulier pour l'obtention de la structure propre d'une matrice polynomiale. D'autre part, le CEP peut être formulé aussi à l'aide des matrices de Sylvester. Les matrices de Sylvester sont dérivées naturellement des matrices polynomiales. Elles s'obtiennent en arrangeant les coefficients de la matrice polynomiale dans une matrice Toeplitz par blocs, cf. l'Exemple 8. Dans (Zúñiga *et al.*, 2003; Zúñiga *et al.*, 2004b; Zúñiga *et al.*, 2006; Zúñiga *et al.*, 2004c; Henrion *et al.*, 2005) on montre que cette approche, appelée par la suite l'approche Toeplitz est très efficace pour le problème du calcul de la structure propre d'une matrice polynomiale, et que c'est une alternative fiable à l'approche des faisceaux.

Les opérations qu'on effectue sur les matrices Toeplitz sont en général des factorisations pour obtenir les rangs, les espaces nuls ou pour résoudre des systèmes d'équations. Pour accomplir ces tâches plusieurs méthodes numériques peuvent être utilisées : la décomposition en valeurs singulières, l'élimination Gaussienne avec pivot, les factorisations orthogonales comme la factorisation QR, ou encore d'autres méthodes qui servent a dévoiler le rang d'une matrice (Chan, 1987). Toutes ces méthodes sont très bien étudiées dans la littérature (Golub *et al.*, 1996) et leur bonnes propriétés numériques sont prouvées (Higham, 1996; Stewart, 1998).

5. Remarques finales

La puissance des ordinateurs modernes a permis à la science et à l'ingénierie d'avancer énormément ces dernières décades. En commande, la simulation des problèmes et conception des solutions à l'aide de l'ordinateur sont devenues des pratiques communes. Pratiques qui, naturellement, sont possibles grâce au développement de logiciels.

Ce développement de logiciels comprend plusieurs aspects dont la plupart sont transparents aux utilisateurs. Dans cet article nous avons étudié les aspects numériques : la limitation en la précision avec laquelle les nombres réels sont représentés par l'ordinateur pose des problèmes numériques qui doivent être traîtés soigneusement pour obtenir des résultats satisfaisants.

L'analyse numérique est devenue la discipline qui permet d'étudier ces problèmes numériques et les algorithmes pour les résoudre. Ainsi les sciences et disciplines mathématiques ont développé leurs branches d'analyse numérique dont l'algèbre linéaire numérique est une des plus riches.

Ce développement de l'algèbre linéaire numérique a permis de résoudre beaucoup de problèmes en commande, formulés via l'approche espace d'état. Actuellement l'abondance de logiciels de commande assistée par ordinateurs (CACSD) a rendu l'approche espace d'état la méthode de prédilection quand des résultats numériques sont requis.

D'autres approches à la commande comme l'approche polynomiale, très développées au niveau théorique, sont plus limitées numériquement. Certains logiciels de CACSD correspondants sont basés sur le calcul symbolique qui implique un temps d'exécution et une demande en mémoire plus importants. Mais surtout, quand des coefficients inexacts sont traîtés, les résultats obtenus par le calcul symbolique ne sont pas utilisables en pratique (ils peuvent être donnés comme des fractions de nombres avec des centaines de chiffres par exemple).

Quand un problème sur des polynômes ou des matrices polynomiales est numériquement réductible, c'est-à-dire qu'il peut être reformulé comme un problème équivalent sur des matrices et vecteurs constants, l'utilisation des méthodes numériques résulte en des algorithmes plus performants. Dans cet article nous avons mentionné que les opérations élémentaires sur les polynômes, cause de l'instabilité numérique des méthodes polynomiales classiques, peuvent être évitées grâce à cette reformulation du problème original.

L'instabilité des opérations élémentaires est donc remplacée par la stabilité des méthodes numériques de l'algèbre linéaire. La solution du problème constant équivalent est alors traduite en termes du problème polynomial original mais malheureusement, l'exactitude de cette solution n'est pas assurée. Deux aspects affectent cette qualité :

1) la manière par laquelle l'erreur arrière sur le problème constant équivalent est reflétée sur les coefficients du problème polynomial original,

2) la manière par laquelle la solution du problème original répond à cette perturbation sur les données d'entrée.

Les deux aspects ci-dessus impliquent deux analyses basiques : l'analyse de l'erreur et l'analyse de la sensibilité. Ces analyses sont bien comprises dans le cas de l'algèbre linéaire numérique, mais ils ne peuvent pas être appliqués de la même façon aux problèmes non-linéaires comme ceux impliquant les polynômes. La consolidation d'une algèbre non-linéaire numérique reste donc une grande lacune à combler pour que les problèmes numériques sur les polynômes et les matrices polynomiales soient résolus complètement. Des travaux très récents sur l'algèbre polynomiale numérique semblent indiquer que certains problèmes polynomiaux mal posés peuvent être résolus de manière satisfaisante.

Ainsi, nous croyons qu'il est une question de temps pour qu'il existe davantage de logiciels numériques de qualité pour les polynômes et les matrices polynomiales, et pour que le développement des logiciels de CACSD pour l'approche polynomiale soit comparable à celui de l'approche espace d'état. En attendant, l'algèbre polynomiale numérique et la systématisation de méthodes polynomiales se dévoile comme une ligne de recherche très prometteuse.

6. Bibliographie

Abou-Kandil H., Freiling G., Jank G., Ionescu V., *Matrix Riccati Equations in Control and Systems Theory*, Birkhäuser Verlag, Berlin, 2003.

Ackermann J., « Der Entwurf Linearer Regelungssysteme im Zustandraum », *Regelungstechnik und Prozess-Datenverarbeitung*, vol. 7, p. 297-300, 1972.

Anderson E., Bai Z., Bischof C., Blackford S., Demmel J., Dongarra J., Croz J. D., Greenbaum A., Hammarling S., McKenney A., Sorensen D., *LAPACK Users' Guide*, SIAM, Philadelphia, 1999.

Basilio J. C., Moreira M. V., « A robust solution of the generalized polynomial Bezout identity », *Linear Algebra Appl.*, vol. 385, p. 287-303, 2004.

Beelen T. G. J., Veltkamp G. W., « Numerical computation of a coprime factorization of a transfer function matrix », *Systems Control Lett.*, vol. 9, p. 281-288, 1987.

Bini D., Pan V., *Polynomial and Matrix Computations, Vol 1 : Fundamental Algorithms*, Birkhauser, Boston, 1994.

Boettcher P., *Frequently asked questions in newsgroup* `comp.soft-sys.matlab`, MIT, Massachussetts. 2000.

Chabert J. L., *A History of Algorithms – From the Pebble to the Microchip*, Springer-Verlag, Milan, 1999.

Chan T. F., « Rank revealing QR Factorizations », *Linear Algebra Appl.*, vol. 88/89, p. 67-82, 1987.

Cipra B. A., « The best of the 20th century : Editors name top 10 algorithms », *SIAM News*, vol. 33, n° 4, p. 1-2, 2000.

Datta B. N., *Linear Algebra and its Role in Linear Systems Theory*, AMS Contemporary Mathematics Series, Providence, 1985.

Dedieu J. P., Tisseur F., « Perturbation theory for homogeneous polynomial eigenvalue problems », *Linear Algebra Appl.*, vol. 358, p. 71-94, 2003.

Demmel J., Dongarra J., ST-HEC : reliable and scalable software for linear algebra computations on high-end computers, Lapack working note 164, University of California and University of Tennessee, Feb., 2005.

Dongarra J., Sullivan F., « Guest editors introduction to the top 10 algorithms », *Computing in Science and Engineering*, vol. 2, n° 1, p. 22-23, 2000.

Edelman A., Murakami H., « Polynomial roots from companion matrix eigenvalues », *Math. Comput.*, vol. 64, p. 763-776, 1995.

Fan H. Y., Lin W. W., van Dooren P., « A note on optimal scaling of second order polynomial matrices », *SIAM J. Matrix Anal. Appl.*, vol. 26, p. 252-256, 2004.

Gantmacher F. R., *Theory of Matrices*, Chelsea, New York, 1959.

Gleick J., « A bug and a crash. Sometimes a bug is more than a nuisance », *New York Times Magazine*, December 1st, 1996.

Goldberg D., « What every computer scientist should know about floating-point arithmetic », *ACM Computing Surveys*, vol. 23, n° 1, p. 5-48, 1991.

Golub G. H., van Loan C. F., *Matrix Computations*, Johns Hopkins University Press, New York, 1996.

Hadamard J., « Sur les problèmes aux dérivées partielles et leur signification physique », *Princeton University Bulletin*, p. 49-52, 1902.

Henrion D., Reliable Algorithms for Polynomial Matrices, PhD thesis, Institute of Information Theory and Automation, Academy of Sciences of the Czech Republic, Prague, 1998.

Henrion D., Šebek M., « Reliable numerical methods for polynomial matrix triangularization », *IEEE Trans. Automat. Control*, vol. 44, p. 497-508, 1999.

Henrion D., Zúñiga J. C., « Detecting infinite zeros in polynomial matrices », *IEEE Trans. Circuits Systems II*, vol. 52, n° 12, p. 744-745, 2005.

Herget C. J., Laub A. J., « Special Issue on Computer-Aided Design of Control Systems », *IEEE Control Systems Magazine*, 1982.

Higham N. J., *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, 1996.

Higham N. J., Mackey D. S., Tisseur F., The conditioning of linearizations of matrix polynomials, Numerical analysis report no. 456, Manchester Centre for Computational Mathematics, UK, April, 2005.

Higham N., Konstantinov M., Mehrmann V., « Sensitivity of computational control problems », *IEEE Control Systems Magazine*, vol. 24, p. 28-43, 2004.

Hämmerlin G., Hoffmann K. H., *Numerical Mathematics*, Springer-Verlag, New York, 1999.

Hromčík M., Numerical algorithms for polynomial factorizations, PhD thesis, Department of Control Engineering, Czech Technical University in Prague, Czech Rep., 2004.

Hunt K. J., *Polynomial Methods in Optimal Control and Filtering*, IEE Control Engineering Series 49, Peter Peregrinus, London, 1993.

Inria, *Launch of the Scilab Consortium dedicated to scientific computing*, France. 2003.

Ipsen I. C. F., « Accurate eigenvalues for fast trains », *SIAM News*, Nov., 2004.

Kailath T., *Linear Systems*, Prentice Hall, Englewood Cliffs, 1980.

Kautsky J., Nichols N. K., van Dooren P. M., « Robust pole assignment in linear state feedback », *Internat. J. Control*, vol. 41, n° 5, p. 1129-1155, 1985.

Kraffer F., « Polynomial matrix to state space conversion without polynomial reduction », *IEEE Mediterranean Symposium on Control and Automation*, Chania, Greece, 1996.

Kraffer F., State-space algorithms for polynomial matrix operations, Amarilla no. 135, Department of Electrical Engineering of CINVESTAV, Mexico, D.F., 1998.

Kučera V., *Discrete Linear Control : The Polynomial Equation Approach*, John Wiley and Sons, Chichester, 1979.

Kučera V., « Diophantine equations in control–a survey », *Automatica*, vol. 29, p. 1361-1375, 1993.

Laub A. J., « Numerical linear algebra aspects of control design computations », *IEEE Trans. Automat. Control*, vol. 30, p. 97-108, 1985.

Lawson C. L., Hanson R. J., Kincaid D. R., Krogh K. T., « Basic Linear Algebra Subprograms for FORTRAN usage », *ACM Trans. Math. Software*, vol. 5, p. 308-323, 1979.

Lemonnier D., van Dooren P., « Optimal scaling of companion pencils for the QZ-algorithm », *SIAM Conference on Applied Linear Algebra*, Williamsburg, VA, 2003.

Lemonnier D., van Dooren P., « Optimal scaling of block companion pencils », *International Symposium on Mathematical Theory of Networks and Systems*, Leuven, Belgium, 2004.

Mackey D. S., Mackey N., Mehl C., Mehrmann V., Vector spaces of linearizations for matrix polynomials, Numerical analysis report no. 464, Manchester Centre for Computational Mathematics, UK, April, 2005.

Mathworks, *Matlab, Version 7, Release 14*, Natick, Pennsylvania. 2004.

Miminis G., Paige C. C., « An algorithm for pole assignment of time invariant linear systems », *Internat. J. Control*, vol. 35, p. 341-345, 1982.

Miminis G., Paige C. C., « A direct algorithm for pole assignment of time invariant multi-input linear systems using state feedback », *Automatica*, vol. 24, p. 343-356, 1988.

Moler C. B., Floating points. IEEE standard unifies arithmetic model, Cleve's corner, Matlab newsletter News & Notes, Fall, 1996.

Moler C. B., Stewart G. W., « An algorithm for generalized matrix eigenvalue problems », *SIAM J. Numer. Anal.*, vol. 10, p. 241-256, 1973.

Moler C. B., van Loan C. F., « Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later », *SIAM Review*, vol. 45, n° 1, p. 801-837, 2003.

Munro N., *Symbolic methods in control and system analysis and design*, IEE Control Enginnering Series 56, London, UK, 1999.

Neumann J. V., Goldstine H. H., « Numerical inverting of matrices of high order », *Bull. Amer. Math. Soc.*, vol. 53, p. 1021-1099, 1947.

Osborne E. E., « On pre-conditioning of matrices », *J. ACM*, vol. 7, p. 338-345, 1960.

Overton M. L., *Numerical computation with IEEE floating point arithmetic*, SIAM, Philadelphia, 2001.

Parlett B. N., Reinsch C., « Balancing a matrix for calculation of eigenvalues and eigenvectors », *Numer. Math.*, vol. 13, p. 293-304, 1969.

Petkov P., Christov N., Konstantinov M., *Computational Methods for Linear Control Systems*, Prentice Hall, Englewood Cliffs, 1991.

Polderman J. W., Willems J. C., *Introduction to Mathematical Systems Theory : a Behavioral Approach*, Springer-Verlag, 1998.

PolyX, *The Polynomial Toolbox for Matlab, Version 2.5*, Prague, Czech Republic. 2000.

Press W. H., Flannery B. P., Teukolsky S. A., *Numerical Recipes in Fortran : The Art of Scientific Computing*, Cambridge University Press, 1992.

Rice J. R., « A theory of condition », *SIAM J. Numer. Anal.*, vol. 3, n° 2, p. 287-310, 1966.

Rosenbrock H., *State-Space and Multivariable Theory*, Wiley, New York, 1970.

Skeel R., « Roundoff error and the Patriot missile », *SIAM News*, vol. 25, n° 4, p. 11, 1992.

Stefanidis P., Papliński A. P., Gibbard M. J., *Numerical operations with polynomial matrices : Application to multi-variable dynamic compensator design*, Lecture Notes in Control and Information Sciences, 171, Springer Verlag, New York, 1992.

Stefanovski J., « Polynomial J-spectral factorization in minimal state-space », *Automatica*, vol. 39, p. 1893-1901, 2003.

Stetter H. J., « Numerical polynomial algebra : concepts and algorithms », *Asian Technology Conference in Matemathics*, Chiang Mai, Thailand, 2000.

Stetter H. J., *Numerical Polynomial Algebra*, SIAM, Philadelphia, 2004.

Stewart G. W., *Matrix Algorithms*, SIAM, Philadelphia, 1998.

Tisseur F., « Backward error and condition of polynomial eigenvalue problems », *Linear Algebra Appl.*, vol. 309, p. 339-361, 2000.

Tisseur F., Meerbergen K., « The quadratic eigenvalue problem », *SIAM Review*, vol. 43, p. 235-286, 2001.

Turing A. M., « Rounding-off errors in matrix processes », *Quart. J. Mech. Appl. Math.*, vol. 1, p. 287-308, 1948.

van Dooren P. M., « The computation of Kronecker's canonical form of a singular pencil », *Linear Algebra Appl.*, vol. 27, p. 103-140, 1979.

van Dooren P. M., « The basics of developing numerical algorithms », *IEEE Control Systems Magazine*, vol. 24, p. 18-27, 2004.

van Dooren P. M., Dewilde P., « The eigenstructure of an arbitrary polynomial matrix : computational aspects », *Linear Algebra Appl.*, vol. 50, p. 545-580, 1983.

Varga A., « A Descriptor System Toolbox for Maltab », *IEEE International Symposium on Computer Aided Control System Design*, Anchorage, Alaska, 2000.

Varga A., « Computation of least order solutions of linear rational equations », *International Symposium on Mathematical Theory of Networks and Systems*, Leuven, Belgium, 2004.

Waterloo-Maple, *Maple, version 9.5*, Waterloo, Canada. 2005.

Wilkinson J. H., *The Algebraic Eigenvalue Problem*, Oxford University Press, 1965.

Wilkinson J. H., *Rounding Errors in Algebraic Processes*, Dover, New York, 1994.

Wolfram-Research, *Mathematica, version 5.1*, Champaign, Illinois. 2005.

Zeng Z., « Computing multiple roots of inexact polynomials », *Math. Comput.*, vol. 64, p. 869-903, 2005.

Zhang H., « Numerical condition of polynomials in different forms », *Electron. Trans. Numer. Anal.*, vol. 12, p. 66-87, 2001.

Zúñiga J. C., Henrion D., « Comparison of algorithms for computing infinite structural indices of polynomial matrices », *European Control Conference*, Cambridge, UK, 2003.

Zúñiga J. C., Henrion D., Block Toeplitz algorithms for polynomial matrix null-space computation, Research report no. 04669, LAAS-CNRS, Toulouse, France, December, 2004a. Submitted to SIAM J. Matrix Anal. Appl.

Zúñiga J. C., Henrion D., « Block Toeplitz methods in polynomial matrix computations », *International Symposium on Mathematical Theory of Networks and Systems*, 2004b.

Zúñiga J. C., Henrion D., « On the application of displacement structure methods to obtain null-spaces of polynomial matrices », *IEEE Conference on Decision and Control*, Paradise Island, Bahamas, 2004c.

Zúñiga J. C., Henrion D., « Numerical stability of block Toeplitz algorithms in polynomial matrix computations », *IFAC World Congress on Automatic Control*, Prague, Czech Republic, 2005.

Zúñiga J. C., Henrion D., « A Toeplitz algorithm for the polynomial J-spectral factorization », *Automatica*, vol. 42, p. 1085-1093, 2006.