Université de Lille 1 SCIENCES ET TECHNOLOGIES

CRIStAL Centre de Recherche en Informatique, Signal et Automatique de Lille

Bases creuses en algèbre linéaire exacte et simplification algorithmique de modèles biologiques

THÈSE

présentée et soutenue publiquement le 11 juillet 2016

pour l'obtention du

Doctorat de l'Université de Lille

(mention informatique)

par

Alexandre Temperville

Composition du jury

Rapporteurs :	François Fages	INRIA Paris-Saclay
	Ovidiu Radulescu	LIRMM, Université de Montpellier 2
Examinateurs :	Cédric Lhoussaine	CRIStAL, Université de Lille
	Claude-Pierre Jeannerod	LIP, ENS de Lyon
Directeur :	François Boulier	CRIStAL, Université de Lille
Co-directeur :	François Lemaire	CRIStAL, Université de Lille

Université de Lille - Sciences et Technologies
Centre de Recherche en Informatique, Signal et Automatique de Lille
U.F.R. d'I.E.E.A. - Bât. M3 - 59655 VILLENEUVE D'ASCQ CEDEX

Mis en page avec la classe thesul.

Je dédie cette thèse
à mon fidèle pc.
Oui, à ce cher Perceval !
Il n'est pas très futé,
mais doué pour les calculs.

Remerciements

Au moment où j'écris ces lignes, je me remémore avec nostalgie mes parcours scolaires, professionnels et personnels, qui m'ont permis d'être qui je suis et d'avoir l'immense honneur d'arriver là où j'en suis aujourd'hui. J'ai l'immense privilège et la joie de vous présenter à travers cette thèse aujourd'hui la concrétisation de mes quatre années de travaux passionnés durant mon doctorat, en mêlant informatique, mathématiques et biologie. Il a été pour moi l'occasion d'évoluer en terme de compétences professionnelles et de connaissances scientifiques. Je réalise que de nombreuses personnes – collègues, enseignants et proches – ont contribué de près ou de loin à ce que j'en arrive à réaliser ce doctorat et à rédiger cette thèse. Cette rubrique leur est dédiée.

Mes premiers remerciements vont à mon directeur de thèse, François Boulier, pour m'avoir proposé et permis de réaliser cette thèse sous sa direction, pour la confiance et ses encouragements qu'il m'a donnés durant ces quatre années de recherche. Je le remercie également pour ses précieux conseils et remarques, qu'ils soient tournés sur les TP de "Structure de Données" dans lesquels je suis intervenu avec lui ou sur le déroulement du doctorat et la rédaction de cette thèse.

Je remercie ensuite fortement mon encadrant, François Lemaire, qui m'a suivi avec bienveillance et patience durant ces quatre années de doctorat, et avec qui j'ai énormément appris. J'ai apprécié passer du temps avec lui à éclaircir mes travaux, à profiter de son sens du détail d'une extrême et juste rigueur afin de les mettre en avant avec clarté et pédagogie. Je le remercie également pour ses nombreuses relectures et aides à produire les deux publications que nous avons co-publiées ainsi que cette thèse.

Je remercie François Fages et Ovidiu Radulescu d'avoir accepté d'être rapporteurs de ma thèse ainsi que leurs nombreuses remarques qui ont contribué à améliorer ce document. Je remercie également Cédric Lhoussaine et Claude-Pierre Jeannerod d'avoir accepté de faire partie des membres du jury et de s'être fortement intéressés à mes travaux.

Je remercie l'ensemble de l'équipe CFHP pour m'avoir accueilli chaleureusement et dans laquelle j'ai pris beaucoup de plaisir à travailler. J'y ai découvert le calcul formel et pu profité de nombreuses discussions enrichissantes. Je remercie notamment Alexandre Sedoglavic, avec lequel j'ai bénéficié de nombreux conseils sur la pédagogie en intervenant dans les TD et TP de "Pratique du langage C" et "Shell et langage de script". Je remercie également Marie-Émilie Voge pour nos discussions sur la programmation linéaire et pour m'avoir permis de clarifier des points techniques à ce sujet intervenant dans cette thèse. Je remercie ensuite Charles Bouillaguet avec lequel j'ai pu concevoir des TP de "Systèmes et Traitement Répartis" ainsi que pour les nombreuses discussions scientifiques que nous avons eu. Enfin, je remercie Adrien Poteaux pour ses remarques pertinentes qui ont contribué à améliorer mes exposés et Claire Delaplace pour sa jovialité. Je souhaite également remercier d'anciens membres de l'équipe : Paul-Émile Boutoille

et Ainhoa Aparicio Monforte pour les nombreuses discussions philosophiques et scientifiques que nous avons eues, Valentin Owczarek pour m'avoir expliqué l'utilisation de Git, et enfin Aurélien Greuet pour m'avoir montré que oui, on peut imprimer un document avec juste une simple ligne de commande.

Je remercie également Céline Kuttler et Jean-Christophe Routier, pour lesquels je suis intervenu en TD et TP de "Base de Données" et "Programmation Orientée Objet" respectivement. Ils m'ont permis, grâce à leur disponibilité et leurs nombreuses explications, d'enseigner ces disciplines efficacement.

Je remercie pour finir ma famille et mes amis. Je ne serais pas ce que je suis aujourd'hui sans le soutien imperturbable de mes parents Jean-Pierre Temperville et Françoise Vermeulen ainsi que de mes frères David et Jérôme durant l'ensemble de mes études, ils ont toujours été là pour moi et je leur en serai éternellement reconnaissant. Je tiens à remercier mes amis, anciens camarades de classe, binômes de travail et/ou amis de longue date – tout particulièrement Sébastien Capelle, Amandine Vanehuin, Benoît Vanehuin, Mathilde Dequirez, Julien Ellart, François Clermont, Joseph Laurent, Joffrey Vanhollemeersch, Pauline Vanhollemeersch, Julien Nachtergaele, Hélène Seghers, Pauline Duchenne, Antoine Defossez et Mickaël Alvarez – qui, malgré la distance pour certains d'entre eux, ont toujours été présents à mes côtés et soutenu dans mon parcours. Enfin, je remercie ma tendre et attentionnée compagne Leslie Robette, qui m'a soutenu jour après jour et encouragé sans relâche à donner le meilleur de moi-même durant la rédaction de ma thèse, et qui a supporté mes divagations scientifiques nocturnes à parler de backslashs ou encore de retour chariot.

Sommaire

Remerciements **iii**

Introduction générale **ix**

1 Contexte biologique **1**

1.1 Modélisation par réactions chimiques 2

1.1.1 Systèmes de réactions chimiques 2

1.1.2 Matrice de stœchiométrie 4

1.1.3 Dynamique déterministe 4

1.1.4 Base de données BioModels 6

1.2 Bases de lois de conservation 8

1.2.1 Lois de conservation 8

1.2.2 Caractère creux et positivité 10

1.2.3 Bases de lois de conservation 11

1.2.4 Complexité du calcul de bases les plus creuses 13

1.2.5 Calcul de bases creuses 14

1.2.6 P-invariants 14

2 Calcul de bases les plus creuses **17**

2.1 Introduction 18

2.2 Bases les plus creuses 20

2.2.1 Définitions 20

2.2.2 Approche gloutonne 20

2.2.3 Structure des bases les plus creuses ligne-équivalentes 21

2.3 Algorithme CSB 21

2.3.1 Tâches 21

2.3.2 Algorithme CSB 23

2.3.3 Algorithme EnhanceBasis 23

2.3.4 Algorithme BasisToSolvedTask 24

2.3.5 Algorithme EnhanceBasisUsingSolvedTask . . . 25
2.3.6 Algorithme IsTask . . . 26
2.3.7 Algorithme NextVector . . . 29
2.4 Complexité et implantation . . . 29
2.4.1 Complexité . . . 29
2.4.2 Implantation . . . 30
2.4.3 Améliorations . . . 31
2.5 Expérimentations . . . 31
2.5.1 Temps d'exécution des différentes versions de CSB . . . 32
2.5.2 Gain du pré-traitement avec CSBprime . . . 33
2.5.3 Gain du découpage par blocs et influence de certains paramètres . . . 34
2.5.4 Taux de réduction . . . 36

3 Calcul de bases creuses par la programmation linéaire 37
3.1 Introduction . . . 38
3.2 Mise en place d'un programme linéaire . . . 39
3.2.1 Système non-linéaire NLP . . . 39
3.2.2 Équivalence de NLP avec un programme linéaire LP . . . 41
3.2.3 Sommets du polyèdre défini par LP . . . 44
3.3 Simplexe et algorithmes . . . 47
3.3.1 Programme linéaire sous forme canonique faisable . . . 47
3.3.2 Algorithme InitialisationSimplexe . . . 49
3.3.3 Algorithme CSBsimplexe . . . 51
3.3.4 Complexité et implantation . . . 52
3.4 Expérimentations . . . 53
3.4.1 Liste de fonctions d'objectifs $\mathcal{L}_1$ . . . 53
3.4.2 Liste de fonctions d'objectifs $\mathcal{L}_2$ . . . 54
3.4.3 Pré-traitement par CSBsimplexe avec $\mathcal{L}_1$ pour CSB . . . 55
3.4.4 Calculs sur des modèles trop volumineux pour CSB . . . 56

4 Simplification de systèmes paramétriques d'équations 57
4.1 Introduction . . . 58
4.2 Fractions rationnelles . . . 59
4.2.1 Représentation matricielle d'une fraction . . . 59
4.2.2 Application monomiale . . . 60
4.2.3 Action d'une application monomiale . . . 60
4.2.4 Base d'un espace vectoriel modulo un espace vectoriel . . . 61

4.2.5 L'algorithme CSBmodulo . . . 61
4.2.6 Preuve de CSBmodulo . . . 63
4.2.7 L'algorithme getSparsestFraction . . . 65
4.3 Sommes de fractions rationnelles . . . 69
4.3.1 Représentation matricielle d'une somme de fractions . . . 69
4.3.2 Action d'une application monomiale . . . 70
4.3.3 Algorithme getSparsestSumOfFractions . . . 71
4.3.4 Application à la simplification de systèmes d'équations . . . 72
4.3.5 Complexité et implantation . . . 73

5 Reverse Engineering de modèles **75**
5.1 Introduction . . . 75
5.2 Modèle BIOMD253 . . . 76
5.2.1 Mise en place d'un programme linéaire . . . 77
5.2.2 Expérimentations sur différents objectifs . . . 78
5.3 Modèle BIOMD050 . . . 79

Conclusion **83**

Annexes **85**

A Temps d'exécution des différentes versions de CSB **85**

B Nombre de nonzéros après application d'algorithmes matriciels **87**

C Paramètres ayant une influence sur le temps d'exécution de la version v_2' **89**

D Résultats des expérimentations avec CSBsimplexe **91**

Index **93**

Bibliographie **95**

Sommaire

Introduction générale

Cette thèse s'inscrit dans une série de travaux de recherche de l'équipe CFHP autour de la modélisation en biologie. En particulier les nouveaux algorithmes présentés dans cette thèse contribuent à la thématique générale de la simplification des modèles biologiques : le calcul de bases creuses de lois de conservation, la simplification des systèmes d'équations différentielles paramétriques, fréquents en modélisation, et le *reverse engineering* des modèles.

On se concentre sur la modélisation par systèmes de réactions chimiques, qui décrivent la transformation de certaines *espèces chimiques*, appelées *réactants*, en d'autres espèces, appelées *produits*. Cette description doit être complétée par une dynamique précise, qui permet de simuler l'évolution des espèces dans le temps. Selon [49], huit dynamiques sont envisageables : l'espace d'état peut être continu ou discret (C ou D) ; le temps peut être continu ou discret (C ou D) ; la détermination peut être stochastique ou déterministe (S ou D). Les systèmes dynamiques modélisés par équations différentielles sont du type CCD. Les systèmes stochastiques simulables par l'algorithme de Gillespie sont du type DCS.

Pour des raisons historiques, l'équipe CFHP a beaucoup travaillé la simplification des modèles présentés par équations différentielles ordinaires non linéaires [8, 9, 14, 12, 10, 11, 13, 16, 40, 39].

Les méthodes décrites dans cette thèse sont plus générales que celles étudiées jusqu'ici, parce qu'elles s'intéressent aux simplifications qui peuvent se déduire de la matrice de stœchiométrie définie par le système de réactions chimiques. En effet, les lois de conservation linéaires qui s'en déduisent sont communes aux huit dynamiques, et pas seulement à la dynamique différentielle déterministe.

Cette simplification, qui ne met en œuvre que des méthodes d'algèbre linéaire exacte, présente le gros avantage de permettre le traitement de modèles de taille réaliste. En particulier, les méthodes de la thèse sont validées sur des exemples tirés de la base de données BioModels [1].

La thèse présente trois nouveaux algorithmes : l'algorithme CSB et deux variantes. L'algorithme CSB est un algorithme d'algèbre linéaire exacte. Il calcule une base la plus creuse parmi toutes les bases d'un espace vectoriel. Typiquement, on l'applique à l'ensemble des lois de conservation déduites de la matrice de stœchiométrie. La sortie de l'algorithme est garantie mais au prix d'une complexité en temps, exponentielle en le nombre d'espèces dans le pire des cas.

La première variante, appelée CSBsimplexe, résout le même problème que CSB via plusieurs résolutions de programmes linéaires en variables réelles (l'algorithme du simplexe). Les réels sont toutefois représentés par des nombres rationnels exacts. Cet algorithme semble plus efficace que CSB sur des problèmes de grande taille puisqu'il nous a permis de traiter des exemples issus de la base de données BioModels, qui étaient inaccessibles à notre implantation de CSB (les modèles BIOMD205 et BIOMD457). Cet algorithme a toutefois deux inconvénients : la base qu'il calcule n'est pas nécessairement complète ; même dans le cas où elle l'est, il se peut qu'elle ne soit pas la plus creuse possible.

La deuxième variante, appelée CSBmodulo, résout le même problème que CSB mais modulo un espace vectoriel F. Cet algorithme a été mis au point avec l'objectif de simplifier des fractions rationnelles. Informellement, l'espace F permet de coder une propriété des fractions : leur valeur est inchangée après multiplication du numérateur et du dénominateur par une même quantité.

Une importante contribution de cette thèse concerne la simplification des modèles différentiels déterministes dont les membres droits sont des fractions rationnelles. De tels modèles sont très fréquents en modélisation en biologie (échanges de type Michaelis-Menten, par exemple). Des travaux plus anciens, menés dans l'équipe CFHP, ont produit des algorithmes de réduction exacte qui exploitent la présence de symétries des équations différentielles de type dilatation. Ces algorithmes de réduction diminuent le nombre des paramètres présents dans les équations. Notre contribution est complémentaire à ces méthodes puisqu'elle permet d'accentuer le caractère creux des équations. Il semble qu'elle ait aussi souvent pour effet de réduire leurs degrés.

L'idée consiste à chercher des changements de variables monomiaux, par calcul de bases les plus creuses sur des espaces vectoriels définis par les exposants des variables du système. On la valide sur un exemple qui avait précédemment fait l'objet de nombreuses études [13, 10, 56, 8].

Dans notre contexte, l'idée du *reverse engineering* — directement inspirée des motivations exposées dans [24] — est de tenter de deviner le système de réactions chimiques à partir du modèle mathématique (par exemple, à partir du système différentiel déterministe). Un bon logiciel dédié au reverse engineering de modèles biologiques aurait un impact fort sur la communauté des modélisateurs puisqu'il aiderait à valider les bases de données de modèles.

Notre contribution tient à une idée : l'absence de certaines lois de conservation (à coefficients uniquement positifs) dans les bases les plus creuses suggère une espèce chimique manquante, soit oubliée lors de la modélisation, soit supprimée par le modélisateur après une étape de réduction de modèle non documentée.

On valide cette idée sur deux exemples intéressants, tirés de la base de données BioModels [1]. Cette contribution pourrait donner lieu à une extension du logiciel Nicotine [54] et de [24]. Nos travaux suggèrent donc des pistes prometteuses pour le futur.

Le chapitre 1 présente brièvement quelques aspects de la modélisation en biologie, notamment sur les lois de conservation, directement utiles à cette thèse. Le chapitre 2 présente l'algorithme CSB, qui permet de calculer des bases les plus creuses. Le chapitre 3 présente l'algorithme CSBsimplexe basé sur la programmation linéaire, qui permet de calculer des bases creuses (mais pas nécessairement les plus creuses). Le chapitre 4 détaille l'application à la simplification de fractions à l'aide de l'algorithme CSBmodulo. Enfin, le chapitre 5 expose l'application au reverse engineering de modèles par suggestion d'ajout d'espèces dans les réactions.

Table des figures

1.1 BIOMD361 – Système de réactions chimiques et graphe de réactions 3
1.2 BIOMD361 – Matrice de stœchiométrie 4
1.3 Modèle simple SBML d'un système de réactions décrivant une dégradation enzymatique 9
1.4 Dégradation enzymatique 11
1.5 Modèles dont les nombres de lois de conservation indépendantes entre Nicotine et CSB diffèrent 15

2.1 Temps d'exécution des différentes versions de CSB sur un échantillon de modèles 32
2.2 Nombre de nonzéros après application des algorithmes matriciels de CSBprime . . 33
2.3 Diagramme - Nombre de nonzéros après application des algorithmes matriciels de CSBprime 34
2.4 Paramètres ayant une influence sur le temps d'exécution de la version v_2' 35
2.5 Nombre de bases donnant certaines proportions $\rho = \frac{\mathcal{N}(B')}{\mathcal{N}(B)}$ 36

3.1 Résultats après application de CSBsimplexe sur $\mathcal{L}_1$ sur un échantillon de modèles 54
3.2 Résultats après application de CSBsimplexe sur $\mathcal{L}_2$ sur un échantillon de modèles 55
3.3 Temps d'exécution de calculs d'une base la plus creuse avec CSBsimplexe puis CSB 56
3.4 Gros modèles 56

4.1 Circuit électrique, composé de deux résistances R_1, R_2 et d'un condensateur C. . 68

A.1 Temps d'exécution des différentes versions de CSB 85
A.2 Temps d'exécution des différentes versions de CSB (suite) 86

B.1 Nombre de nonzéros après application de CSBprime 87
B.2 Nombre de nonzéros après application de CSBprime (suite) 88

C.1 Paramètres ayant une influence sur le temps d'exécution de la version v_2' 89
C.2 Paramètres ayant une influence sur le temps d'exécution de la version v_2' (suite) . 90

D.1 Résultats des expérimentations avec CSBsimplexe 91
D.2 Résultats des expérimentations avec CSBsimplexe (suite) 92

Chapitre 1

Contexte biologique

Sommaire

1.1 Modélisation par réactions chimiques 2
1.1.1 Systèmes de réactions chimiques 2
1.1.2 Matrice de stœchiométrie 4
1.1.3 Dynamique déterministe 4
1.1.4 Base de données BioModels 6
1.2 Bases de lois de conservation 8
1.2.1 Lois de conservation 8
1.2.2 Caractère creux et positivité 10
1.2.3 Bases de lois de conservation 11
1.2.4 Complexité du calcul de bases les plus creuses 13
1.2.5 Calcul de bases creuses 14
1.2.6 P-invariants 14

L'utilisation de modèles en biologie est courante. Ils permettent de représenter des processus biologiques que les scientifiques cherchent à décrire et à comprendre [57, page 3]. Les modèles que nous étudions dans cette thèse sont des systèmes de réactions chimiques mettant en jeu des espèces. Nous manipulerons ces systèmes de réactions chimiques principalement pour les deux raisons suivantes. Premièrement, nous utilisons dans cette thèse la base de données *BioModels* [1, 37, 42], libre d'accès, et contenant des modèles représentés sous différents formats dont le format *SBML* que nous utiliserons (voir section 1.1.4). Deuxièmement, l'étude des systèmes de réactions chimiques est une des thématiques de l'équipe de recherche *CFHP* dans laquelle j'ai effectué mes travaux de thèse. Ces systèmes de réactions chimiques peuvent être étudiés de façon qualitative ou de façon quantitative. Différentes approches peuvent être considérées. Le choix d'une approche dépend des données connues (équations, paramètres, fluctuations...), de ce que l'on veut obtenir (évolution des concentrations d'espèces ou flux d'espèces au cours du temps par exemple) et de la façon dont on souhaite procéder.

L'approche par *Équations Différentielles Ordinaires (EDO)* est très répandue. Elle permet d'étudier quantitativement les modèles et est déterministe. Elle permet de décrire un modèle par un *système d'équations différentielles*. L'approche par EDO nécessite que les concentrations d'espèces ne varient pas en fonction de l'espace, et que les concentrations d'espèces soient suffisamment grandes.

Si certaines espèces sont en faible quantité ou si le milieu n'est pas bien mélangé ("well stired" en anglais), l'approche par processus stochastiques est pertinente [2]. À partir d'un processus

stochastique, on peut réaliser des *Simulations Stochastiques* (SS), qui en général sont différentes même lorsque les conditions initiales sont fixées. Cette approche permet d'étudier des modèles quantitativement. L'algorithme de Gillespie [27, 28, 29] décrit comment procéder pour réaliser des simulations stochastiques.

Nous considérons dans toute cette thèse que les *compartiments* (volumes finis dans lesquels les espèces intéragissent) sont généralement homogènes. Si on considère que la position des espèces dans les compartiments n'est pas homogène, une approche par EDP (Équations aux Dérivées Partielles) est requise. Cette approche est également déterministe et permet d'étudier des modèles quantitativement. Elle complexifie le modèle sans nécessairement le rendre plus cohérent.

On peut également étudier les modèles de façon qualitative avec par exemple l'approche par réseaux de booléens [57, Section 7.4 Boolean Networks]. Dans cette approche, on considère par exemple qu'une espèce est soit présente, soit absente, sans considération de quantité. On peut trouver différentes approches dans [57, page 14].

1.1 Modélisation par réactions chimiques

1.1.1 Systèmes de réactions chimiques

Le vocabulaire introduit dans cette partie ainsi que la définition 1.1 sont largement inspirés de [15].

La *réaction chimique* $A \rightharpoonup B$ décrit la transformation d'une *espèce* A en une espèce B. L'espèce A est un *réactant* tandis que l'espèce B est un *produit* . On peut enrichir la notation d'une réaction par un paramètre k, appelé *constante de réaction*, caractérisant la vitesse de transformation. Les sens du paramètre k et de son unité dépendent du contexte. Toutefois, plus k est grand, plus la réaction a tendance à se réaliser. On note dans ce cas la réaction chimique $A \xrightarrow{k} B$. Parfois, on remplace la constante de réaction par une vitesse de réaction.

La réaction chimique $A + B \rightharpoonup C$ décrit une transformation plus complexe que l'on peut interpréter comme suit : lorsqu'une molécule A rencontre une molécule B, elles peuvent réagir pour former une molécule C. La réaction chimique $A + B \rightleftharpoons C$ décrit une réaction *réversible*. Elle peut être vue comme une paire de réactions. De la droite vers la gauche, cette réaction peut être interprétée comme suit : chaque molécule C peut se transformer en formant à la fois une molécule A et une molécule B. On peut également annoter des constantes de réactions pour chacun des deux sens avec $A + B \underset{k^-}{\overset{k}{\rightleftharpoons}} C$.

Dans toute cette thèse, pour une réaction chimique r_i, on notera k_i la constante de la réaction r_i dans son sens direct (de la gauche vers la droite) et k_i^- la constante de réaction de son sens indirect (de la droite vers la gauche) s'il existe (i.e. si la réaction est réversible).

La définition suivante est inspirée de [15] :

Définition 1.1. *Soient $A_1, \ldots, A_n$ des espèces. Toute réaction chimique* irréversible *peut s'écrire sous la forme d'une combinaison linéaire d'espèces $r : \alpha_1 A_1 + \cdots + \alpha_n A_n \rightharpoonup \beta_1 A_1 + \cdots + \beta_n A_n$ où $\alpha_i, \beta_i \in \mathbb{R}_+$ pour tout $i \in [\![1, n]\!]$. Toute réaction chimique* réversible *peut s'écrire sous la forme d'une combinaison linéaire d'espèces*

$$r : \alpha_1 A_1 + \cdots + \alpha_n A_n \rightleftharpoons \beta_1 A_1 + \cdots + \beta_n A_n \tag{1.1}$$

ou sous la forme d'un couple de réactions irréversibles

$$\begin{cases} r^+ : \alpha_1 A_1 + \cdots + \alpha_n A_n \rightharpoonup \beta_1 A_1 + \cdots + \beta_n A_n \\ r^- : \beta_1 A_1 + \cdots + \beta_n A_n \rightharpoonup \alpha_1 A_1 + \cdots + \alpha_n A_n \end{cases}$$

où $\alpha_i, \beta_i \in \mathbb{R}_+$ pour tout $i \in [\![1, n]\!]$. Sous cette deuxième forme, la notation r^+ représente la réaction réversible r dans son sens direct (i.e. de la gauche vers la droite) tandis que la notation r^- représente r dans son sens indirect (i.e. de la droite vers la gauche).

En principe, toutes les réactions chimiques sont réversibles. Cependant, certaines réactions peuvent être considérées pratiquement irréversibles. On peut donc exclure de l'analyse les sens qui ne se produisent (quasiment) jamais. Les notations r^+ et r^- peuvent être utilisées pour désigner les sens d'une réaction réversible r écrite sous la forme de l'équation (1.1).

Remarque 1.1. *En pratique, dans les algorithmes que nous présenterons dans les chapitres suivants, les coefficients α_i, β_i apparaissant dans les réactions chimiques (écrites comme à la définition 1.1) sont dans $\mathbb{Q}_+$.*

Exemple 1.1. *Dans le contexte de coagulation du sang, le modèle BIOMD361, issu de la base de données BioModels (voir section 1.1.4), décrit des réactions qui se produisent lors de dommages vasculaires, déclenchées par le complexe $VIIa\_TF$, créé lorsqu'un contact se produit entre une membrane et le plasma (voir [48]). Le complexe $VIIa\_TF$ est une enzyme qui catalyse les réactions chimiques de ce modèle. Pour des raisons de simplicité, nous renommons les espèces intervenant dans ces réactions de la façon suivante : $X \to S$ (substrat), $VIIa\_TF \to E$ (enzyme), $TFPI \to I$ (inhibiteur), $Xa \to P$ (produit). Les notations ES, EP, PI et PIE représentent ici des complexes formés par les espèces précédentes. Par exemple, PIE représente l'espèce $Xa\_TFPI\_VIIa\_TF$.*

$$\begin{cases} r_1 : S + E & \underset{k_1^-}{\overset{k_1}{\rightleftharpoons}} ES \\ r_2 : ES & \overset{k_2}{\rightharpoonup} EP \\ r_3 : EP & \underset{k_3^-}{\overset{k_3}{\rightleftharpoons}} E + P \\ r_4 : P + I & \underset{k_4^-}{\overset{k_4}{\rightleftharpoons}} PI \\ r_5 : E + PI & \underset{k_5^-}{\overset{k_5}{\rightleftharpoons}} PIE \end{cases},$$

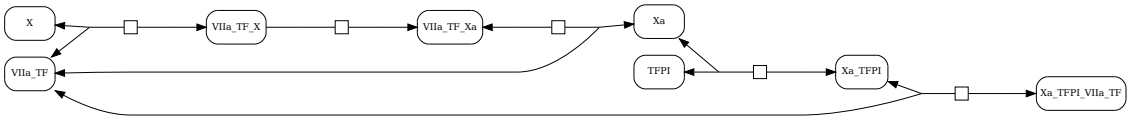

FIGURE 1.1 – BIOMD361 – Système de réactions chimiques et graphe de réactions

La figure 1.1 présente le système de réactions chimiques décrivant le modèle ainsi que son graphe de réactions associé, où chacune des boîtes représente une réaction. Les graphes de réactions de BioModels sont au format SBGN[1].

1. Le graphe de réactions obtenu depuis le site de BioModels par l'option "View Reaction Graph" n'est pas 100% compatible avec le format SBGN d'après le site de BioModels[1].

1.1.2 Matrice de stœchiométrie

Définition 1.2. *Soit un système de réactions chimiques. Soient $A_1, \dots, A_n$ des espèces. Soient q réactions chimiques r_j, où $j \in [\![1, q]\!]$, écrites comme dans la définition 1.1 sous la forme*

$$r_j : \alpha_{1,j} A_1 + \cdots + \alpha_{n,j} A_n \rightharpoonup \beta_{1,j} A_1 + \cdots + \beta_{n,j} A_n.$$

On appelle matrice de stœchiométrie *la matrice M de dimension $n \times q$ telle que chaque ligne correspond à une espèce, chaque colonne correspond à une réaction et tout élément $\mu_{i,j}$ de M représente le coefficient stœchiométrique de l'espèce A_i dans la réaction r_j. Il est égal à $\mu_{i,j} = \beta_{i,j} - \alpha_{i,j}$.*

Dans une matrice de stœchiométrie, l'ordre des lignes et l'ordre des colonnes dépend de l'ordre dans lequel on classe les espèces et les réactions. Plusieurs matrices de stœchiométrie peuvent donc représenter le même système de réactions chimiques. Ces matrices de stœchiométrie dépendent du système de réactions chimiques mais pas de la dynamique utilisée.

Dans [57, pages 76-77], il est écrit que si une espèce est présente en grande quantité dans un compartiment, elle peut ne pas apparaître dans le modèle, et n'est donc pas représentée dans la matrice de stœchiométrie. En effet, sa concentration n'évolue quasiment pas, vu qu'elle est très élevée. Par exemple, l'eau, souvent présente en abondance, n'apparaît pas dans les modèles. Si une espèce se trouve à l'extérieur du compartiment, bien qu'elle intéragisse avec les espèces du compartiment, elle n'apparaît pas dans le modèle. On appelle une telle espèce une *espèce externe*. D'après [57, pages 76-77], "les espèces externes sont des sources ou des puits" (i.e. elles alimentent le compartiment de l'extérieur, ou disparaissent hors du compartiment). Ainsi, une réaction $A \rightharpoonup \emptyset$ ne signifie donc pas que A disparaît mais que A sort du système ou bien que A se transforme en une espèce présente en grande quantité.

Exemple 1.2. *Reprenons le modèle BIOMD361 décrit dans l'exemple 1.1 par la figure 1.1. En considérant chaque réaction réversible r_i comme un couple de réactions irréversible r_i^+ et r_i^-, comme dans la définition 1.1, on construit la matrice de stœchiométrie M de dimension 8×9 à la figure 1.2.*

$$\begin{array}{c} \begin{array}{ccccccccc} r_1^+ & r_1^- & r_2 & r_3^+ & r_3^- & r_4^+ & r_4^- & r_5^+ & r_5^- \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \end{array} \\ \left[\begin{array}{ccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 & -1 & -1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & -1 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 \end{array}\right] \end{array} \begin{array}{l} \leftarrow PIE \\ \leftarrow P \\ \leftarrow S \\ \leftarrow EP \\ \leftarrow ES \\ \leftarrow I \\ \leftarrow E \\ \leftarrow PI \end{array}$$

FIGURE 1.2 – BIOMD361 – Matrice de stœchiométrie

1.1.3 Dynamique déterministe

Dans toute cette thèse, la concentration d'une espèce A à l'instant t est notée $[A](t)$, voire $A(t)$ ou A, lorsque le contexte est clair.

Dans les approches par EDO que nous étudions, l'évolution des concentrations des espèces par rapport au temps peut être décrite par le système d'équations différentielles [30] suivant :

$$\frac{dc(t)}{dt} = M \cdot v(t) \tag{1.2}$$

où $c(t)$ est un vecteur de dimension n représentant les concentrations des espèces à l'instant t, M est la matrice de stœchiométrie, $v(t)$ est un vecteur de dimension q représentant les vitesses de réactions. Ces vitesses de réactions sont parfois connues de manière empirique (expérimentalement) et parfois déduites de la loi d'action de masse. Dans ce second cas, une constante de réaction doit être spécifiée. Dans une réaction $r_i : \alpha_1 A_1 + \cdots + \alpha_n A_n \xrightarrow{k_i} \beta_1 A_1 + \cdots + \beta_n A_n$, si la *loi d'action de masse* se vérifie, alors la vitesse de réaction $v_i(t)$ est le produit entre la constante de réaction k_i et les concentrations d'espèces des réactants (avec pour exposants leurs coefficients α_i) i.e. $v_i(t) = k_i \prod_{j=1}^{n} A_j(t)^{\alpha_j}$.

Exemple 1.3. *Reprenons le modèle BIOMD361 décrit dans l'exemple 1.1 par la figure 1.1. On fixe le vecteur des espèces*

$$c(t) = \begin{pmatrix} PIE(t) \\ P(t) \\ S(t) \\ EP(t) \\ ES(t) \\ I(t) \\ E(t) \\ PI(t) \end{pmatrix}, \tag{1.3}$$

puis on ordonne les réactions comme dans l'exemple 1.2. Le modèle BIOMD361 définit le vecteur de vitesses de réactions en utilisant la loi d'action de masse, on a donc

$$v(t) = \begin{pmatrix} k_1 S(t) E(t) \\ k_1^- ES(t) \\ k_2 ES(t) \\ k_3 EP(t) \\ k_3^- E(t) P(t) \\ k_4 P(t) I(t) \\ k_4^- PI(t) \\ k_5 E(t) PI(t) \\ k_5^- PIE(t) \end{pmatrix}. \tag{1.4}$$

En réutilisant la matrice de stœchiométrie M définie à la figure 1.2, le système d'équations

différentielles (1.2) *pour le modèle BIOMD361 est de la forme :*

$$\begin{cases} PIE'(t) & = k_5 E(t) PI(t) - k_5^- PIE(t) \\ P'(t) & = k_3 EP(t) - k_3^- E(t) P(t) - k_4 P(t) I(t) + k_4^- PI(t) \\ S'(t) & = -k_1 S(t) E(t) + k_1^- ES(t) \\ EP'(t) & = k_2 ES(t) - k_3 EP(t) + k_3^- E(t) P(t) \\ ES'(t) & = k_1 S(t) E(t) - k_1^- ES(t) - k_2 ES(t) \\ I'(t) & = -k_4 P(t) I(t) + k_4^- PI(t) \\ E'(t) & = -k_1 S(t) E(t) + k_1^- ES(t) + k_3 EP(t) - k_3^- E(t) P(t) - k_5 E(t) PI(t) + k_5^- PIE(t) \\ PI'(t) & = k_4 P(t) I(t) - k_4^- PI(t) - k_5 E(t) PI(t) + k_5^- PIE(t) \end{cases} \tag{1.5}$$

On remarque que les colonnes de M représentant les réactions r_i^+ et r_i^- sont naturellement de signes opposés donc linéairement dépendantes (voir figure 1.2). Par la suite, nous ne ferons pas apparaître les colonnes des réactions r_i^- dans la matrice de stœchiométrie. Dans l'exemple 1.3, si on réécrit $v(t)$ sous la forme

$$v(t) = \begin{pmatrix} k_1 S(t) E(t) - k_1^- ES(t) \\ k_2 ES(t) \\ k_3 EP(t) - k_3^- E(t) P(t) \\ k_4 P(t) I(t) - k_4^- PI(t) \\ k_5 E(t) PI(t) - k_5^- PIE(t) \end{pmatrix}$$

et la matrice de stœchiométrie M sous la forme

$$\begin{array}{c} \begin{array}{ccccc} r_1 & r_2 & r_3 & r_4 & r_5 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 \\ -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ -1 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix} \end{array} \begin{array}{l} \leftarrow PIE \\ \leftarrow P \\ \leftarrow S \\ \leftarrow EP \\ \leftarrow ES \\ \leftarrow I \\ \leftarrow E \\ \leftarrow PI \end{array}$$

alors le système d'équations différentielles défini par l'équation (1.5) est inchangé. Cela revient d'ailleurs à considérer chaque réaction réversible comme une seule réaction ; c'est de cette façon que SBML décrit les modèles (voir exemple 1.4).

Tout comme le système d'équations différentielles (1.5), il est possible que les systèmes d'équations différentielles décrivant les modèles soient non linéaires. Cela complique fortement leur résolution exacte directe.

1.1.4 Base de données BioModels

Cette section doit beaucoup à [42, 17, 35, 37]. Le grand nombre de données disponibles dans le domaine de la biologie a poussé la communauté scientifique à utiliser des dépôts centralisés. Ces

dépôts permettent à la communauté scientifique de pouvoir communiquer, échanger et mettre à jour les modèles sur lesquels de nombreux utilisateurs sont susceptibles de travailler.

La base de données BioModels est un dépôt en ligne [1] de modèles mathématiques décrivant des processus biologiques. Elle permet de stocker, visualiser, extraire et analyser des modèles biologiques, qu'ils soient issus de publications ou non. Elle est devenue une référence reconnue pour les systèmes biologiques. BioModels propose également un certain nombre de fonctionnalités. Le concept de BioModels et ses fonctionnalités sont décrits dans les articles suivants [42, 17, 35, 37].

La plupart de ces modèles sont annotés par des termes issus d'un vocabulaire spécifique et reliés à des ressources externes facilitant leur lisibilité et leur réutilisation. Les modèles peuvent être déposés ou extraits de BioModels sous différents formats : SBML, BioPAX, SciLab, Octave, XPP, VCML, REF/XML.

Le 16 février 2016, cette base de données contient exactement 1450 modèles publiés dans la littérature scientifique et 143070 modèles automatiquement générés à partir de ressources "pathway".

Parmi les 1450 modèles issus de la littérature scientifique, 583 ont été *réarrangés* manuellement (on parle en anglais de "curated models"). Cela signifie que ces 583 modèles permettent de reproduire les résultats des expérimentations décrites dans les publications qui leur sont associées. Ce sont sur ces derniers modèles que nos expérimentations des chapitres suivants porteront.

Les modèles issus de BioModels que nous utiliserons dans cette thèse seront manipulés à partir du format SBML (System Biology Markup Language). C'est un langage qui est basé sur le langage XML, il permet de représenter des modèles de processus biologiques. Ce langage est libre et ouvert. Il s'intègre dans de nombreux autres langages, tels que le C, le C++, Java, Python, Mathematica... De nombreux logiciels, plus de 200 en 2010, incorporent SBML. Il est le langage de référence pour la représentation de modèles biologiques. Il est proposé selon différentes spécifications (niveaux) [34, 43], qui se distinguent par les fonctionnalités qu'elles proposent. Par exemple, le niveau 3, le plus récent, enrichit les fonctionnalités du niveau 2 (pour plus de détails sur ces niveaux, consulter la documentation SBML sur [1]).

Un modèle en SBML (pour le niveau 2 [43]) peut contenir les éléments suivants, décrits dans [57, Section 17.3 - The SBML Representation of Models] :

- un ou plusieurs *compartiments* (espaces de dimensions finies dans lesquels se situent les espèces) ;
- des *espèces* (elles représentent en fait des concentrations ou des quantités de matières présentes) ;
- des *réactions* (transformations, transports modifiant les états des espèces) caractérisées par la stœchiométrie de leurs produits et réactants, et optionnellement par des équations donnant leurs vitesses de réaction ;
- des *paramètres* (comme des constantes de réaction) ;
- des *définitions d'unités* (noms d'unités utilisées pour les quantités dans le modèle) ;
- des *règles* (expressions mathématiques) que l'on ajoute aux équations du modèle. Elles permettent par exemple d'associer des valeurs aux paramètres, des contraintes entre des quantités, ... ;
- des *fonctions mathématiques* que l'on peut utiliser à la place d'expressions répétées dans les formules du modèle ;
- des *évènements* (ensembles de formules mathématiques évaluées à un instant spécifique au cours de l'évolution du système) ;
- ...

Exemple 1.4. *On s'intéresse dans cet exemple au modèle décrivant la dégradation enzymatique. Elle est définie par le système de réactions*

$$\begin{cases} S + E \underset{k_r}{\overset{k_1}{\rightleftharpoons}} ES \\ ES \overset{k_2}{\rightharpoonup} E + P \end{cases} \tag{1.6}$$

avec les espèces E (enzyme), S (substrat), P (produit), ES (complexe intermédiaire) et les constantes de réactions k_1, k_r, k_2.

La figure 1.3 donne une représentation SBML minimaliste du système de réactions (1.6)*. Cette représentation spécifie un compartiment Cell dans lequel les espèces S, E, ES et P se situent ainsi que leurs quantités initiales. La réaction réversible $S + E \rightleftharpoons ES$, est définie avec une loi cinétique donnant sa vitesse de réaction globale par $k_1 S(t)E(t) - k_r ES(t)$. La réaction irréversible $ES \rightharpoonup E + P$ est définie avec une loi cinétique donnant sa vitesse de réaction $k_2 ES(t)$. Des unités par défaut sont utilisées lorsqu'aucune unité n'est définie dans un modèle SBML pour les quantités. L'unité par défaut des quantités de matières est la mole, celle des volumes le litre. Les paramètres k_1, k_r, k_2 ainsi que leurs valeurs respectives* 3, 6, 9 *figurent également dans ce modèle.*

Bien entendu, on peut étoffer ce modèle en y rajoutant toute information pertinente. On pourrait également l'écrire autrement (considérer 3 réactions réversibles, changer les unités, ...). Si on considérait deux réactions irréversibles pour la première réaction du système (1.6)*, on aurait deux lois cinétiques définissant les vitesses de réactions des sens direct et indirect de la première réaction : $k_1 \times S(t) \times E(t)$ et $k_r \times ES(t)$. Dans ce cas, le vecteur $v(t)$ aurait un élément de plus et la matrice de stœchiométrie une colonne de plus, dans la même idée que l'exemple 1.3. Cependant, les modèles SBML sont généralement écrits de sorte qu'une réaction réversible soit considérée comme une seule réaction.*

La simplicité du formalisme de SBML permet de modéliser un large champ de phénomènes biologiques, comme la signalisation cellulaire, la régulation de gènes, etc.

1.2 Bases de lois de conservation

1.2.1 Lois de conservation

Les lois de conservation sont des relations qui peuvent être utilisées pour réduire le nombre d'espèces (inconnues) dans les systèmes d'équations différentielles. D'après [59], prendre en compte des lois de conservation permet au simulateur de réduire la taille d'un modèle. Dans cette thèse, les lois de conservation sont utilisées pour les modèles déterministes parce qu'elles se déduisent de la matrice de stœchiométrie, mais elles sont également valables pour des modèles stochastiques.

Définition 1.3. *Soient $A_1, \ldots, A_n$ des espèces. Une* loi de conservation linéaire *est une somme pondérée de concentrations d'espèces qui reste constante au cours du temps. Les* lois de conservation linéaires *s'écrivent sous la forme*

$$\alpha_1 A_1(t) + \cdots + \alpha_n A_n(t) = constante$$

où $\alpha_i \in \mathbb{R}$ pour tout i. On les représente par le vecteur $\begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix}$.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1">
  <model id="EnzymeKinetics">
    <listOfCompartments>
      <compartment id="Cell" size="1"/>
    </listOfCompartments>
    <listOfSpecies>
      <species id="S" compartment="Cell" initialAmount="1" boundaryCondition="true"/>
      <species id="E" compartment="Cell" initialAmount="1"/>
      <species id="ES" compartment="Cell" initialAmount="0.01"/>
      <species id="P" compartment="Cell" initialAmount="0.01" boundaryCondition="true"/>
    </listOfSpecies>
    <listOfReactions>
      <reaction id="Reaction1">
        <listOfReactants>
          <speciesReference species="S"/>
          <speciesReference species="E"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="ES"/>
        </listOfProducts>
        <kineticLaw>
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply> <minus/>
            <apply> <times/> <ci> k_1 </ci> <ci> S </ci> <ci> E </ci> </apply>
            <apply> <times/> <ci> k_r </ci> <ci> ES </ci> </apply>
            </apply>
          </math>
          <listOfParameters>
            <parameter id="k_1" value="3"/>
            <parameter id="k_r" value="6"/>
          </listOfParameters>
        </kineticLaw>
      </reaction>
      <reaction id="Reaction2" reversible="false">
        <listOfReactants>
          <speciesReference species="ES"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="E"/>
          <speciesReference species="P"/>
        </listOfProducts>
        <kineticLaw>
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply> <times/> <ci> k_2 </ci> <ci> ES </ci> </apply>
          </math>
          <listOfParameters>
            <parameter id="k_2" value="9"/>
          </listOfParameters>
        </kineticLaw>
      </reaction>
    </listOfReactions>
  </model>
</sbml>
```

FIGURE 1.3 – Modèle simple SBML d'un système de réactions décrivant une dégradation enzymatique

Nous nous intéressons dans cette thèse aux lois de conservation linéaires. Avec la définition 1.3, toute combinaison linéaire de lois de conservation est également une loi de conservation. C'est également vrai pour des lois non linéaires et pour des produits de lois de conservation.

Des lois de conservation du type $A(t) + B(t) = constante$ apparaissent fréquemment dans l'étude de modèles biologiques. Cette loi de conservation signifie que lorsque A est consommée, B est produit, conservant la somme des concentrations de ces deux espèces constante. Cela se traduit, dans la matrice de stœchiométrie du modèle où cette loi apparaît, par une dépendance linéaire entre les deux lignes représentant ces deux espèces.

Soit $c(t) = {}^t(A_1, \dots, A_n)$ le vecteur des concentrations d'espèces A_i. Soit y un vecteur de dimension n tel que ${}^t y c'(t) = 0$. Alors la définition 1.3 implique que y représente une loi de conservation. En effet, en intégrant ${}^t y c'(t) = 0$, on obtient ${}^t y c(t) = constante$.

On en déduit donc, à l'aide du système (1.2) la proposition suivante :

Proposition 1.1. *Soient M une matrice de stœchiométrie de dimension $n \times q$ et y un vecteur de dimension n. Si ${}^t M y = 0$, alors y est une loi de conservation.*

Démonstration. Soit un vecteur y tel que ${}^t M y = 0$. Alors ${}^t y M = 0$. On multiplie à gauche par ${}^t y$ l'égalité du système (1.2) et on obtient : ${}^t y c'(t) = {}^t y M v(t) = 0$. Ainsi, on conclut que y est une loi de conservation. □

La démonstration de cette proposition se base sur une dynamique déterministe. Cependant, la proposition est également valable pour toutes les dynamiques. Cette proposition permet de trouver des lois de conservation du modèle grâce à l'algèbre linéaire, via la matrice de stœchiométrie, sans se soucier du sytème d'équations différentielles initial. Nous utiliserons cette proposition pour donner un ensemble de lois de conservation indépendantes. La réciproque de cette proposition n'est cependant pas vraie, comme l'illustre l'exemple suivant.

Exemple 1.5. *Soient A, B, C des espèces. Soit* $\begin{cases} A \xrightarrow{k} B \\ A \xrightarrow{k} C \end{cases}$ *un système de réactions et* $M = \begin{pmatrix} -1 & -1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$ *sa matrice de stœchiométrie. Les deux réactions de ce système ont la même constante de réaction k. Si on utilise l'approche par EDO avec l'équation (1.2), on montre que $B - C = constante$, i.e. ${}^t y = \begin{pmatrix} 0 & 1 & -1 \end{pmatrix}$ est une loi de conservation. Toutefois, on a ${}^t M y \neq 0$. Il s'agit d'un cas très particulier, qui à notre connaissance ne se produit pas en pratique. Comme il y a la même constante de réaction k et le même réactant A, les deux réactions sont en fait couplées et peuvent s'écrire comme une seule réaction $A \xrightarrow{k} \frac{B}{2} + \frac{C}{2}$ dans le cas de l'approche par équations différentielles.*

1.2.2 Caractère creux et positivité

Exemple 1.6. *Reprenons le modèle décrivant la dégradation enzymatique donné dans l'exemple 1.4. La figure 1.4 donne le système de réactions chimiques qui définit la dégradation enzymatique, le vecteur des concentrations d'espèces $c(t)$, la matrice de stœchiométrie M et le vecteur des vitesses de réactions $v(t)$.*

Ce modèle admet, par exemple, pour lois de conservation $E + ES = constante$ et $S + ES + P = constante$. On peut obtenir les vecteurs représentant ces lois en calculant le noyau de ${}^t M$. On peut les additionner/soustraire pour en déduire d'autres lois de conservation, par exemple $E + 2ES + S + P = constante$ et $E - S - P = constante$. Les lois de conservation $E + ES =$

$$\begin{cases} S + E \underset{k_r}{\overset{k_1}{\rightleftharpoons}} ES \\ ES \xrightarrow{k_2} E + P \end{cases}, c(t) = \begin{pmatrix} S(t) \\ E(t) \\ ES(t) \\ P(t) \end{pmatrix}, M = \begin{pmatrix} -1 & 0 \\ -1 & 1 \\ 1 & -1 \\ 0 & 1 \end{pmatrix}, v(t) = \begin{pmatrix} k_1 S(t)E(t) - k_r ES(t) \\ k_2 ES(t) \end{pmatrix}.$$

FIGURE 1.4 – Dégradation enzymatique

constante et $S + ES + P =$ constante semblent moins artificielles que les lois de conservation $E + 2ES + S + P =$ constante et $E - S - P =$ constante. En effet, la loi $E + ES =$ constante correspond à la conservation de l'enzyme E, tandis que $S + ES + P =$ constante correspond à la conservation du substrat S (conservé sous la forme S, ES ou P).

Il est raisonnable de se dire qu'une "bonne" loi de conservation devrait avoir de nombreux coefficients nuls, i.e. être creuse, et avoir le moins de coefficients négatifs possible.

Caractère creux. Concernant la propriété d'être creuse, on préférera manipuler des lois creuses plutôt que des lois denses, puisque des lois creuses seront plus courtes et donc plus simples à lire. De plus, une loi de conservation creuse peut également être utile lorsque l'on effectue des substitutions dans des équations différentielles pour préserver le caractère creux des équations.

Par exemple, si on a un système creux $\dot{X} = F(X)$, où X est un vecteur d'espèces $X_1, X_2, \ldots$, on peut utiliser une loi de conservation faisant intervenir X_1, par exemple $X_1 + X_5 - X_8 = c_0$ (avec $c_0 \in \mathbb{R}$) pour éliminer la variable X_1 en la substituant par une expression en les autres X_i ($c_0 - X_5 + X_8$ pour cet exemple). Par conséquent, une loi de conservation creuse sera plus susceptible de préserver le caractère creux des équations différentielles.

Positivité. Concernant les coefficients négatifs, nous pensons que les lois de conservation avec des coefficients positifs sont plus susceptibles de représenter une conservation de matière. Considérons par exemple une loi de conservation $X_2 + 2X_3 + X_6 = c_1$ avec $c_1 \geq 0$. Comme les concentrations d'espèces sont toujours positives, on peut déduire de la loi de conservation que $0 \leq X_2 \leq c_1$, $0 \leq X_3 \leq c_1/2$ et $0 \leq X_6 \leq c_1$. Si on considère au contraire une loi de conservation comme $X_3 - X_4 =$ *constante*, X_3 et X_4 peuvent ne pas être bornés. Quand toutes les espèces sont bornées, on peut calculer exactement les espérances des espèces par une technique proche de [60, Second Order Degradation].

De plus, les lois de conservation dont les coefficients sont uniquement positifs ont du sens dans [30, Section 3.1.3 - Conserved Moieties] : elles peuvent définir ce que l'on nomme en anglais des "conserved moieties" (voir aussi [20]).

Caractère creux et positivité. Ces deux critères sont parfois impossibles à satisfaire ensemble. Par exemple, si on dispose des deux lois de conservation $X_1 + X_2 + X_3$ et $X_2 + X_3 + X_4$, alors la différence $X_1 - X_4$ est plus creuse que chacune des deux lois mais implique un coefficient négatif. De plus, dans certains exemples particuliers, il n'y a pas de lois de conservation avec des coefficients uniquement positifs (comme dans $A + B \rightharpoonup \emptyset$ qui admet uniquement $A - B =$ *constante* comme loi de conservation).

1.2.3 Bases de lois de conservation

La proposition 1.1 induit que tout vecteur du noyau de la transposée d'une matrice de stœchiométrie est une loi de conservation. On peut donc définir une base de ce noyau afin d'avoir un ensemble de lois de conservation indépendantes pour le système de réactions chimiques considéré.

De même que dans la section 1.2.2, il paraît pertinent d'obtenir des bases de lois de conservation creuses et avec peu de coefficients négatifs.

Exemple 1.7. *Reprenons le modèle BIOMD361 décrit dans l'exemple 1.1 et sa matrice de stœchiométrie M définie par la figure 1.2. Une base de lois de conservation du modèle s'obtient en calculant une base du noyau de ${}^tM$. La fonction* NullSpace *du paquetage LinearAlgebra du logiciel de calcul formel* Maple *permet de calculer le noyau d'une matrice. Les vecteurs de base calculés (avec* Maple*) sont ensuite stockés dans une matrice ligne par ligne pour former la matrice B suivante :*

$$B = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & -1 & -1 & -1 & -1 & 1 & 0 & 0 \end{pmatrix}. \tag{1.7}$$

Chaque ligne de B représente une loi de conservation. La matrice B représente donc les trois lois de conservation suivantes :

$$\begin{cases} PIE(t) + P(t) + S(t) + EP(t) + ES(t) + PI(t) & = constante \\ PIE(t) + EP(t) + ES(t) + E(t) & = constante \cdot \\ -P(t) - S(t) - EP(t) - ES(t) + I(t) & = constante \end{cases} \tag{1.8}$$

Ces lois sont plus complexes que celles de la dégradation enzymatique dans l'exemple 1.6. Il est naturel de se demander si leur interprétation a un sens et si on peut obtenir des lois de conservation plus simples (courtes et à coefficients positifs) en effectuant des combinaisons linéaires entre elles (i.e. entre les lignes de la matrice B).

Le calcul direct d'une base de lois de conservation par le noyau ne donne pas nécessairement une base creuse avec des coefficients uniquement positifs. La base calculée dans l'exemple 1.7 montre en effet cela. Par ailleurs, nous verrons dans le chapitre 2 que cette base peut être rendue plus creuse par un procédé algorithmique détaillé dans la section 2.1.

Ce paragraphe doit beaucoup à [59]. Le calcul de lois de conservation à partir de logiciels d'analyse de réseaux biochimiques donne souvent des calculs imprécis d'après [59], car ces logiciels se basent généralement sur du calcul numérique. Par exemple, le logiciel COPASI [32, 45] permet de réaliser des simulations stochastiques, déterministes ou hybrides (en utilisant à la fois des méthodes stochastiques et déterministes) de processus biologiques par diverses méthodes numériques détaillées dans [32]. Pour des systèmes de grandes tailles, toujours d'après [59], les calculs réalisés par la plupart des logiciels en biologie des systèmes sont imprécis, notamment pour le calcul de lois de conservation. L'arithmétique sur les entiers peut résoudre ce problème de précision, mais les calculs peuvent être lents pour des systèmes de grande taille. Le logiciel dont l'algorithme est donné dans [59], faisant partie de la structure logicielle SBW (Systems Biology Workbench) [5], propose une méthode basée sur la décomposition QR de Householder pour calculer des bases de lois de conservation. La décomposition QR de Householder est numériquement plus stable que les méthodes numériques (souvent issues de l'algorithme de Gauss-Jordan) utilisées par les autres logiciels.

La Décomposition en Valeurs Singulières (SVD) est l'approche recommandée dans [51] pour traiter des systèmes de grandes tailles et en extraire des bases de lois de conservation. Cependant, [59] explique que cette approche est malgré tout imprécise pour des systèmes de grandes tailles comme celui du réseau métabolique de la bactérie *Eschericha coli* [59, Figure 1].

Nous avons présenté les méthodes de [51] et [59] permettant de calculer des bases de lois de conservation par du calcul numérique. Pour calculer des bases lois de conservation, nous utilisons de l'arithmétique exacte sur les entiers et sur les rationnels dans les chapitres 2 et 3.

Nous avons discuté de méthodes permettant de calculer des bases de lois de conservation. La section suivante expose la complexité du calcul de bases les plus creuses.

1.2.4 Complexité du calcul de bases les plus creuses

Nous verrons dans cette section que le calcul d'une base de lois de conservation la plus creuse est NP-difficile. Par base la plus creuse, nous entendons une base de vecteurs contenant le petit nombre de coefficients non nuls possible. Un algorithme théorique est présenté dans [18, Algorithm 2.1] pour calculer une base la plus creuse possible sans pour autant préciser comment trouver les vecteurs qui la composent. Le théorème suivant est issu de [18, Theorem 3.2] :

Théorème 1.1. *Soit A une matrice réelle. Étant donné un entier positif k, il est NP-difficile de trouver une base de* $\mathrm{Ker}(A)$ *avec un nombre de coefficients non nuls inférieur ou égal à k.*

Bien que l'article [18] suppose que la matrice A est à coefficients réels, nous pensons que le théorème 1.1 est également vrai dans le cas où la matrice A est à coefficients rationnels. En effet, sa démonstration repose sur le calcul de circuits d'un graphe construit à partir de A.

Proposition 1.2. *Soit un système de réactions chimiques à coefficients stœchiométriques rationnels. Il est NP-difficile de calculer une base de lois de conservation la plus creuse.*

Démonstration. Soit A une matrice à coefficients rationnels de dimension $q \times n$. On peut construire un système de q réactions chimiques portant sur n espèces de la manière suivante :

- $\alpha_{i,j} = -A_{j,i}$ et $\beta_{i,j} = 0$ si $A_{j,i} < 0$;
- $\alpha_{i,j} = 0$ et $\beta_{i,j} = A_{j,i}$ si $A_{j,i} > 0$;
- $\alpha_{i,j} = 0$ et $\beta_{i,j} = 0$ si $A_{j,i} = 0$.

On vérifie aisément que ${}^tA$ est la matrice de stœchiométrie du système ainsi construit. Par conséquent, toute base la plus creuse du système de réactions chimiques précédent est en particulier une base la plus creuse de $\mathrm{Ker}(A)$. D'après le théorème 1.1, calculer une base la plus creuse de lois de conservation est donc NP-difficile. □

Proposition 1.3. *Soit $B \in \mathbb{Q}^{m\times n}$ une matrice de rang plein selon les lignes. Soit V le sous-espace vectoriel de $\mathbb{Q}^n$ engendré par les lignes de B. Alors, il est NP-difficile de calculer une base la plus creuse de V à partir de B.*

Démonstration. Soit $A \in \mathbb{Q}^{q\times n}$ une matrice. On peut calculer une base de $\mathrm{Ker}(A)$ en temps polynomial. Ainsi, calculer une base la plus creuse de $\mathrm{Ker}(A)$ se réduit en temps polynomial au calcul d'une base la plus creuse d'un espace V engendré par les lignes d'une matrice B de rang plein selon les lignes. Le théorème 1.1 permet de conclure. □

Ainsi, d'après les propositions 1.2 et 1.3, il est au moins aussi difficile [2] de calculer des bases de lois de conservation d'un système de réactions chimiques que de calculer une base la plus creuse d'une matrice quelconque. Toutefois, on peut espérer qu'en pratique les systèmes de réactions chimiques aient une structure particulière qui facilitera les calculs.

2. C'est la même difficulté par des arguments évidents.

1.2.5 Calcul de bases creuses

Cette section présente des méthodes pour calculer des bases creuses. Des algorithmes heuristiques sont proposés dans [19] pour calculer des bases relativement creuses, mais ne garantissent pas que la base calculée soit la plus creuse possible.

Quelques algorithmes usuellement utilisés en algèbre linéaire peuvent parfois, à partir d'une base donnée, produire une base plus creuse, sans garantir qu'elle soit la plus creuse possible. Dans $\mathbb{Q}$, on peut utiliser par exemple l'algorithme de mise sous forme échelon réduite selon les lignes (RREF) ou encore l'algorithme LLL [41]. Dans $\mathbb{Z}$, on peut utiliser l'algorithme de mise sous forme normale de Hermite (HNF).

Dans le contexte des réseaux dans $\mathbb{Z}$ ($\mathbb{Z}$-lattices), [26] introduit et calcule des (pseudo-) bases "courtes" en utilisant l'algorithme LLL [41] et une variante de la forme normale de Hermite. Dans un contexte numérique (i.e. en utilisant des coefficients flottants), il y a des méthodes pour calculer des bases creuses (comme dans [6] où l'algorithme "turnback" calcule une base creuse sous forme de matrice bande d'une grande matrice creuse).

La section suivante présente des invariants de position dans le contexte des réseaux de Petri, qui permettent également de donner des lois de conservation creuses.

1.2.6 P-invariants

Les *P-invariants* (invariants de position), intervenant dans les réseaux de Petri, permettent de calculer des lois de conservation avec des coefficients positifs dont le support est minimal [55]. Le logiciel Nicotine [54] permet de calculer un ensemble de P-invariants d'un modèle décrit par le format SBML.

Exemple 1.8. *Nicotine permet d'obtenir, à parti du format SBML du modèle BIOMD361, les P-invariants suivants :*

$$\begin{cases} ES(t) + EP(t) + S(t) + P(t) + PI(t) + PIE(t) & = constante \\ E(t) + ES(t) + EP(t) + PIE(t) & = constante \\ I(t) + PI(t) + PIE(t) & = constante \end{cases} \tag{1.9}$$

On a trois lois de conservation indépendantes, et donc une base de lois de conservation du modèle BIOMD361, qui diffère uniquement sur la dernière loi de conservation de la base (1.8) *de l'exemple 1.7, cette dernière contenant des coefficients négatifs.*

Les P-invariants obtenus forment une famille génératrice de lois de conservation, mais pas nécessairement libre. De plus, certains modèles n'admettent pas de bases de lois de conservation avec des coefficients uniquement positifs. C'est le cas de l'exemple suivant où il n'y a aucun P-invariant et seulement une loi de conservation qui contient deux coefficients de signes opposés. Cela implique que sur certains systèmes, les P-invariants ne fournissent de base complète.

Exemple 1.9. *Soit le système contenant l'unique réaction $A + B \rightharpoonup \emptyset$. Ce système admet pour unique loi de conservation $A - B = constante$. Cependant, ce système n'admet pas de P-invariant.*

La figure 1.5 montre les modèles de BioModels pour lesquels les P-invariants ne fournissent pas de base complète de lois de conservation, dans le sens où le nombre de P-invariants indépendants est strictement inférieur au nombre de lois de conservation obtenues par calcul du noyau de ${}^tM$.

Modèle	*dim_lois*	*dim_Pinvariants*	*n*
BIOMD050	2	1	14
BIOMD052	3	1	12
BIOMD119	0	8	9
BIOMD223	20	19	84
BIOMD237	5	4	24
BIOMD243	3	1	23
BIOMD253	1	0	6
BIOMD305	4	1	8
BIOMD332	16	15	78
BIOMD333	12	11	54
BIOMD336	5	4	18
BIOMD338	0	1	51
BIOMD339	0	1	51
BIOMD409	2	0	15
BIOMD424	14	12	55
BIOMD457	23	22	163
BIOMD494	4	0	45

FIGURE 1.5 – Modèles dont les nombres de lois de conservation indépendantes entre Nicotine et CSB diffèrent

avec *dim_lois* le nombre de lois de conservation indépendantes trouvée par le calcul de noyau de ${}^tM$ (après avoir retiré les colonnes des espèces considérées constantes), *dim_Pinvariants* le nombre de lois de conservation indépendantes trouvées par Nicotine et n le nombre d'espèces considérées

Remarque 1.2. *Le logiciel Nicotine, donnant des P-invariants, ne donne pas de lois constantes du type $A(t) =$ constante si le modèle SBML définit A comme constante, parce qu'il s'agit d'une loi déjà connue dans le modèle (et donc par Nicotine) tandis que le calcul du noyau de ${}^tM$ le fait. Pour comparer exactement, il faut donc soit les rajouter aux P-invariants donnés par Nicotine, soit les retirer du noyau de ${}^tM$. Nous avons choisi de modifier (pour la figure 1.5 uniquement) les modèles afin de comparer exactement les dimensions des bases obtenues par calcul des P-invariants d'un côté, par calcul du noyau de ${}^tM$ de l'autre. Pour cela, nous avons retiré du noyau de ${}^tM$ les espèces considérées comme constantes (cela revient à retirer des matrices de stœchiométrie M les colonnes d'espèces considérées comme constantes).*

Remarque 1.3. *Puisque les P-invariants sont des lois de conservation à coefficients uniquement positifs [55], on s'attend à ce que dim_lois $\geq$ dim_Pinvariants. Ce n'est pas le cas pour les modèles BIOMD119, BIOMD338 et BIOMD339 [1]. La distinction des espèces et des paramètres n'étant pas claire dans ces 3 modèles, nous n'avons aucune loi de conservation dans ces modèles et plusieurs P-invariants. Ces modèles seront donc écartés de nos raisonnements dans les prochains chapitres, ce qui donne un total de $17 - 3 = 14$ modèles qui n'admettent pas de base de lois de conservation à coefficients positifs.*

Chapitre 2

Calcul de bases les plus creuses

Sommaire

2.1 Introduction 18
2.2 Bases les plus creuses 20
2.2.1 Définitions 20
2.2.2 Approche gloutonne 20
2.2.3 Structure des bases les plus creuses ligne-équivalentes 21
2.3 Algorithme CSB 21
2.3.1 Tâches 21
2.3.2 Algorithme CSB 23
2.3.3 Algorithme EnhanceBasis 23
2.3.4 Algorithme BasisToSolvedTask 24
2.3.5 Algorithme EnhanceBasisUsingSolvedTask 25
2.3.6 Algorithme IsTask 26
2.3.7 Algorithme NextVector 29
2.4 Complexité et implantation 29
2.4.1 Complexité 29
2.4.2 Implantation 30
2.4.3 Améliorations 31
2.5 Expérimentations 31
2.5.1 Temps d'exécution des différentes versions de CSB 32
2.5.2 Gain du pré-traitement avec CSBprime 33
2.5.3 Gain du découpage par blocs et influence de certains paramètres 34
2.5.4 Taux de réduction 36

Ce chapitre a fait l'objet d'une publication à CMSB2014 [38]. Il présente un algorithme glouton permettant de calculer des bases les plus creuses d'espaces vectoriels finis dont on connaît une base. Il est le fruit de la recherche d'un procédé pour calculer des bases de lois de conservation creuses, dont nous avons discuté dans le chapitre 1.

La section 2.1 illustre notre méthode de calcul d'une base la plus creuse à partir du modèle BIOMD361 [48] de la base de données BioModels [1, 48]. Ensuite, nous discutons des bases les plus creuses et leur structure particulière dans la section 2.2 avant de montrer que le théorème 2.2 justifie l'approche gloutonne de notre algorithme. Nous présentons ensuite l'algorithme CSB permettant de calculer des bases les plus creuses dans la section 2.3 en détaillant les algorithmes qui le composent. Puis, nous discutons de sa complexité et de son implantation dans la section

2.4. Enfin, nous terminons ce chapitre par des expérimentations sur la base de donnée BioModels dans la section 2.5 où nous cherchons à calculer des bases de lois de conservation les plus creuses.

2.1 Introduction

Nous illustrons comment calculer une base la plus creuse sur le modèle BIOMD361 [48] de la base de données BioModels [1, 48]. Le détail de ce modèle est donné par l'exemple 1.1. On reprend la base de lois de conservation (1.7)

$$B = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & -1 & -1 & -1 & -1 & 1 & 0 & 0 \end{pmatrix}$$

dont les colonnes représentent les concentrations d'espèces du vecteur (1.3) rappelé ici :

$$c(t) = \begin{pmatrix} PIE(t) \\ P(t) \\ S(t) \\ EP(t) \\ ES(t) \\ I(t) \\ E(t) \\ PI(t) \end{pmatrix}.$$

Notre méthode pour diminuer le nombre de coefficients non nuls consiste à trouver une combinaison linéaire $w = {}^t vB$ de lignes de B telle que w contienne moins de coefficients non nuls que l'une des lignes B_i de B. Si l'on peut trouver une telle combinaison linéaire w, il semble naturel de remplacer la ligne B_i par w afin de diminuer le nombre total de coefficients non nuls dans B. En répétant ce procédé jusqu'à-ce qu'aucune telle combinaison linéaire puisse être trouvée, on obtient une base la plus creuse. Cette approche est gloutonne et est justifiée à la section 2.3.

Cependant, le remplacement d'une ligne de B par w ne devrait être réalisé que si B reste une base. Il s'agit d'une condition qui est réalisée en remplaçant la ligne B_i de B uniquement si $v_i \neq 0$ (cela signifie que l'information contenue dans la ligne B_i sera conservée).

Considérons la combinaison linéaire $w = {}^t vB$ avec ${}^t v = (\alpha, \beta, \gamma)$, ainsi on a

$$w = (\alpha + \beta, \alpha - \gamma, \alpha - \gamma, \alpha + \beta - \gamma, \alpha + \beta - \gamma, \gamma, \beta, \alpha).$$

Le nombre de coefficients non nuls de w dépend clairement des valeurs de α, β et γ. Afin de calculer ce nombre, on considère différents cas où l'on choisit de mettre à 0 ou non chacun des éléments de w. En théorie, si w contient n éléments, il y a 2^n cas à considérer.

Par exemple, si on souhaite que w soit de la forme $(\neq 0, = 0, = 0, = 0, = 0, \neq 0, = 0, \neq 0)$, on peut considérer le système d'équations et d'inéquations suivant :

$$\begin{cases} \alpha + \beta \neq 0 & \text{(colonne 1 de } B\text{)} \\ \alpha - \gamma = 0 & \text{(colonnes 2 et 3 de } B\text{)} \\ \alpha + \beta - \gamma = 0 & \text{(colonnes 4 et 5 de } B\text{)} \\ \gamma \neq 0 & \text{(colonne 6 de } B\text{)} \\ \beta = 0 & \text{(colonne 7 de } B\text{)} \\ \alpha \neq 0 & \text{(colonne 8 de } B\text{).} \end{cases}$$

Ce système admet par exemple pour solution $(\alpha, \beta, \gamma) = (1, 0, 1)$. La combinaison linéaire correspondante $w = {}^tvB$ est $w = B_1 + B_3 = (1, 0, 0, 0, 0, 1, 0, 1)$. Elle contient 3 coefficients non nuls, et a moins de coefficients non nuls que les lignes B_1, B_2 et B_3 (qui contiennent respectivement 6, 4, et 5 coefficients non nuls). Comme w fait intervenir les lignes B_1 et B_3 (i.e. $\alpha \neq 0$ et $\gamma \neq 0$), on peut remplacer soit B_1 soit B_3 par w. On remarque que remplacer B_2 par w nous amènerait à une matrice de rang 2, ce qui impliquerait que B ne serait plus une base. Si on remplace B_1 par w, on obtient une base B' ligne-équivalente (voir définition 2.4) à B :

$$B' = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & -1 & -1 & -1 & -1 & 1 & 0 & 0 \end{pmatrix}.$$

En pratique, on n'énumère pas tous les schémas possibles de coefficients nuls et non nuls pour le vecteur w. On considère au lieu de cela les colonnes de B les unes après les autres, de la gauche vers la droite, et on construit des systèmes d'équations (correspondant aux coefficients nuls de w) et d'inéquations (correspondant aux coefficients non nuls de w). Comme chaque colonne de B nous amène à considérer deux cas, on construit un arbre binaire de systèmes d'équations et d'inéquations. En faisant cela, de nombreuses branches peuvent être coupées avant que toutes les colonnes de B soient traitées. Par exemple, si on essaie d'éliminer les cinq premières colonnes de w, on obtient le système d'équations :

$$\begin{cases} \alpha + \beta = 0 & \text{(colonne 1 de } B\text{)} \\ \alpha - \gamma = 0 & \text{(colonnes 2 et 3 de } B\text{)} \\ \alpha + \beta - \gamma = 0 & \text{(colonnes 4 et 5 de } B\text{)} \end{cases}$$

qui admet pour unique solution la solution $(\alpha, \beta, \gamma) = (0, 0, 0)$.

En réitérant l'idée développée ci-dessus afin de continuer à réduire le nombre de coefficients non nuls dans B', on peut trouver une nouvelle combinaison linéaire $w' = B'_1 - B'_2 - B'_3 = (0, 1, 1, 0, 0, 0, -1, 1)$ qui a moins de coefficients non nuls que B'_3. En remplaçant B'_3 par w, on obtient la base

$$B'' = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}.$$

Finalement, une dernière tentative de réitérer le procédé décrit précédemment montre qu'il n'existe pas de nouvelle combinaison linéaire de lignes permettant d'améliorer la base B''. On conclut donc que B'' est une base la plus creuse possible (ce point sera démontré à la section 2.2.2).

La base B'' contient 11 coefficients non nuls dont un négatif. Si on tente de l'éliminer, il semble naturel de remplacer B''_3 par $B''_3 + B''_2 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$. Modifier ainsi la base B'' la rend moins creuse, elle contient dans ce cas 13 coefficients non nuls. C'est notamment ce que trouve le logiciel Nicotine en calculant des P-invariants dans l'exemple 1.8. Comme l'explique la section 1.2.2, satisfaire à la fois le caractère creux et la positivité n'est pas toujours possible. La base de lois de conservation du modèle BIOMD361 l'illustre.

Pour résumer, notre méthode suit une approche gloutonne en diminuant successivement le nombre de coefficients non nuls d'une base initiale (à chaque étape en changeant une seule ligne), jusqu'à ce que l'on atteigne une base la plus creuse possible. Chaque amélioration parcourt un arbre binaire où chaque nœud est un système d'équations et d'inéquations.

2.2 Bases les plus creuses

2.2.1 Définitions

Dans cette thèse, nous représentons toute base sous la forme d'une matrice de rang plein selon les lignes dans laquelle chaque ligne est un vecteur générateur de la base. Nous parlons de *base* en considérant cette forme matricielle.

Soit M (resp. v) une matrice (resp. un vecteur), on note $\mathcal{N}(M)$ (resp. $\mathcal{N}(v)$) le nombre de coefficients non nuls de M (resp. v). Soit v un vecteur, on note $\mathcal{N}_k(v)$ le nombre de coefficients non nuls parmi les k premiers coefficients de v.

Effectuer des combinaisons linéaires sur les lignes d'une matrice revient à multiplier cette matrice à gauche par une matrice inversible. Comme nous n'effectuerons que des combinaisons linéaires sur les lignes, nous aurons besoin de la définition suivante :

Définition 2.4. *Deux matrices M et M' de mêmes dimensions sont dites* ligne-équivalentes *si il existe une matrice inversible P telle que $M' = PM$.*

Rechercher une base la plus creuse d'une base B revient finalement à trouver une matrice inversible P qui minimise $\mathcal{N}(PB)$.

Définition 2.5. *Une base B' est une* base la plus creuse *si et seulement si pour toute base B ligne-équivalente à B', on vérifie $\mathcal{N}(B') \leq \mathcal{N}(B)$.*

Pour toute base B, il est évident qu'il existe une base la plus creuse B' ligne-équivalente à B. En effet, il suffit de considérer l'ensemble de toutes les bases ligne-équivalentes à B et de choisir parmi elles une matrice B' de sorte que $\mathcal{N}(B')$ soit minimal.

2.2.2 Approche gloutonne

Notre méthode suit une approche gloutonne. À partir d'une base B, on recherche un vecteur v et un indice i tels que $\mathcal{N}({}^tvB) < \mathcal{N}(B_i)$ et $v_i \neq 0$. Si un tel couple (v, i) existe, on peut diminuer le nombre de coefficients non nuls de B en remplaçant la ligne B_i par ${}^tvB$. Puisque la ligne B_i a été remplacée par une combinaison linéaire faisant intervenir B_i (puisque $v_i \neq 0$), le rang de B ne change pas. Quand il n'existe pas de tel couple (v, i), notre méthode s'arrête et nous permet d'affirmer que la base obtenue est une base la plus creuse. Cette dernière affirmation n'est pas triviale, on peut en effet se demander si le minimum obtenu n'est pas un minimum local. Le théorème suivant justifie l'approche gloutonne.

Théorème 2.2. *Une base B n'est pas une base la plus creuse si et seulement si il existe un vecteur v et un indice j tels que $\mathcal{N}({}^tvB) < \mathcal{N}(B_j)$ et $v_j \neq 0$.*

Démonstration. $\underline{\Leftarrow :}$ On considère une matrice base B' de même dimension que B définie par $B'_i = B_i$ pour $i \neq j$ et $B'_j = {}^tvB$. On montre facilement que les matrices B et B' sont ligne-équivalentes. De plus, $\mathcal{N}(B') < \mathcal{N}(B)$, ce qui prouve que B n'est pas une base la plus creuse.

$\underline{\Rightarrow :}$ Supposons que B soit de dimension $m \times n$. Il existe une base B' ligne-équivalente à B telle que $\mathcal{N}(B') < \mathcal{N}(B)$. Quitte à permuter les lignes de B et celles de B', on peut supposer que $\mathcal{N}(B_1) \geq \mathcal{N}(B_2) \geq \cdots \geq \mathcal{N}(B_m)$ et $\mathcal{N}(B'_1) \geq \mathcal{N}(B'_2) \geq \cdots \geq \mathcal{N}(B'_m)$. Comme $\mathcal{N}(B) = \sum_{i=1}^{m} \mathcal{N}(B_i) > \sum_{i=1}^{m} \mathcal{N}(B'_i) = \mathcal{N}(B')$, il existe un indice k tel que $\mathcal{N}(B_k) > \mathcal{N}(B'_k)$. Comme B et B' sont ligne-équivalentes, chaque ligne de B' est une combinaison linéaire des lignes de B.

Si toutes les $m - k + 1$ lignes $B'_k, B'_{k+1}, \ldots, B'_m$ étaient des combinaisons linéaires des $m - k$ lignes $B_{k+1}, \ldots, B_m$, alors B' ne serait pas une matrice de rang plein. Ainsi, il existe un vecteur v

et des indices j, l avec $j \leq k \leq l$ tels que $B'_l = {}^tvB$ avec $v_j \neq 0$. Comme $\mathcal{N}(B'_l) \leq \mathcal{N}(B'_k) < \mathcal{N}(B_k) \leq \mathcal{N}(B_j)$, on a $\mathcal{N}({}^tvB) = \mathcal{N}(B'_l) < \mathcal{N}(B_j)$ avec $v_j \neq 0$. □

Corollaire 2.1. *Soit B une base. Si B n'est pas une base la plus creuse, alors le théorème 2.2 s'applique. De plus, remplacer la ligne B_i par ${}^tvB$ mène à une base ligne-équivalente à B plus creuse. Voir l'algorithme* EnhanceBasis *à la section 2.3.3 pour une implantation de ce corollaire.*

2.2.3 Structure des bases les plus creuses ligne-équivalentes

Dans toute cette thèse, on parlera souvent de *nonzéros* pour désigner des coefficients non nuls (dans des vecteurs ou matrices). Les proposition 2.4 et corollaire 2.2 donnés ci-dessous ne serviront que dans le chapitre 4. Ils mettent en valeur une structure commune à toutes les matrices les plus creuses ligne-équivalentes entre elles.

Proposition 2.4. *Soit une base la plus creuse $\bar{M}$ d'une base M de dimensions $m \times n$. On suppose que les lignes de $\bar{M}$ et celles de M sont triées par nombre croissant de nonzéros. Soit D la matrice définie par $D\bar{M} = M$. Alors $\mathcal{N}(\bar{M}_i) \leq \mathcal{N}(M_i)$ pour tout $i \in [\![1, m]\!]$. De plus, pour tout couple d'entiers (i, j) tels que $1 \leq i, j \leq m$, si $\mathcal{N}(M_i) < \mathcal{N}(\bar{M}_j)$, alors $D_{ij} = 0$.*

Démonstration. Démontrons le premier point et supposons $\mathcal{N}(M_i) < \mathcal{N}(\bar{M}_j)$ pour i et j avec $D_{ij} \neq 0$. Suivant les idées du corollaire 2.1, $\bar{M}$ n'est pas la plus creuse possible puisque $\bar{M}_j$ pourrait être remplacé par la ligne M_i, plus creuse, vu que $M_i = \sum_j D_{ij}\bar{M}_j$ et $D_{ij} \neq 0$.

Démontrons maintenant que $\mathcal{N}(\bar{M}_i) \leq \mathcal{N}(M_i)$ pour tout $i \in [\![1, m]\!]$. Par contradiction, on suppose qu'il existe un indice k tel que $\mathcal{N}(\bar{M}_k) > \mathcal{N}(M_k)$ et $\mathcal{N}(\bar{M}_i) = \mathcal{N}(M_i)$ pour $i \leq k-1$.

Chaque ligne M_i est une combinaison linéaire des lignes de $\bar{M}$. Si chacune des k premières lignes de M_i était une combinaison linéaire des $k-1$ lignes de $\bar{M}$, alors les k premières lignes ne seraient pas linéairement indépendantes, et M ne pourrait pas être de rang plein. Par conséquent, il existe deux indices i, l et un vecteur ligne v tels que $i \leq k \leq l$, $\bar{M}_i = v\bar{M}$ avec $v_l \neq 0$. Puisque $\mathcal{N}(\bar{M}_l) \geq \mathcal{N}(\bar{M}_k) > \mathcal{N}(M_k) \geq \mathcal{N}(M_i)$, la ligne $\bar{M}_l$ peut être rendue plus creuse en la remplaçant par la ligne M_i, plus creuse, en utilisant le théorème 2.2. On aboutit à une contradiction vu que $\bar{M}$ est la plus creuse possible. □

Le corollaire suivant prouve que toutes les bases les plus creuses d'une base fixée partagent la même structure commune.

Corollaire 2.2. *Soient deux bases les plus creuses $\bar{M}$ et M' de la même base M de dimensions $m \times n$. On suppose que les lignes de $\bar{M}$ et M' sont triées par nombre croissant de nonzéros. Soit T la matrice définie par $\bar{M} = TM'$. Alors, pour tout $i \in [\![1, m]\!]$, on a $\mathcal{N}(\bar{M}_i) = \mathcal{N}(M'_i)$. De plus, T est une matrice triangulaire inférieure par blocs, dont les largeurs des blocs correspondent aux largeurs des blocs des lignes de $\bar{M}$ ayant le même nombre de nonzéros.*

Démonstration. Il s'agit d'une conséquence directe de la proposition 2.4 appliquée deux fois : la première fois en considérant que $\bar{M}$ est une base la plus creuse de M', la deuxième fois en considérant que M' est une base la plus creuse de $\bar{M}$. □

2.3 Algorithme CSB

2.3.1 Tâches

Comme expliqué dans la section 2.1, notre méthode construit un arbre binaire de systèmes d'équations et d'inéquations. En pratique, on ne stocke que les feuilles de l'arbre en construction.

Nous introduisons la notion de *tâche* qui, essentiellement, représente une feuille de l'arbre. Afin de couper des branches inutiles aussitôt que possible, une tâche doit également satisfaire les propriétés **LCP** et **IZP** de la définition 2.6 suivante.

Soit v un vecteur de dimension n. La notation $v \not\equiv 0$ signifie que $\forall i \in [\![1, n]\!]$, $v_i \neq 0$.

Définition 2.6. *Soit B une base. Soient A et Λ des matrices ayant m colonnes. Soit $\mathcal{S} : \begin{cases} Ax = 0 \\ \Lambda x \not\equiv 0 \end{cases}$ un système en la variable x. Soit c le nombre de lignes de Λ. Soit k la somme des nombres de lignes de A et de Λ. Une* tâche $t = \text{TASK}[A, \Lambda, c, k]$*, issue de B, est définie comme suit :*

- *l'union des lignes de A et de Λ coïncident avec les k premières colonnes de B (à l'ordre près) ;*
- *le couple (A, Λ) satisfait la propriété **LCP** (Linear Combination Property), i.e. il existe une solution v de $\mathcal{S}$ avec au moins deux coefficients non nuls ;*
- *le quadruplet (A, Λ, c, k) satisfait la propriété **IZP** (Increase Zeros Property), i.e. il existe une solution v de $\mathcal{S}$ et un indice j tels que $v_j \neq 0$ et $\mathcal{N}_k({}^tvB) < \mathcal{N}(B_j)$.*

Proposition 2.5. *Soient $t = \text{TASK}[A, \Lambda, c, k]$ une tâche issue d'une matrice base B et $\mathcal{S}$ un système défini comme à la définition 2.6. Soit $\mathcal{U} = \{i \in [\![1, m]\!], c < \mathcal{N}(B_i)\}$. Alors, on a les propriétés suivantes :*

1. *$A \in \mathbb{Q}^{(k-c)\times m}$ et $\Lambda \in \mathbb{Q}^{c\times m}$ avec $0 \leq c \leq k \leq n$;*
2. *Pour chaque solution v non nulle de $\mathcal{S}$, $\mathcal{N}_k({}^tvB) = c$ i.e. c est le nombre de coefficients non nuls dans les k premiers coefficients de toute solution de $\mathcal{S}$;*
3. *Il existe une solution v de $\mathcal{S}$ et un indice $j \in \mathcal{U}$ tels que $v_j \neq 0$ et $c < \mathcal{N}(B_j)$.*

Démonstration. 1. Trivial.

2. On prend une solution v non nulle de $\mathcal{S}$ et on considère $w = {}^tvB$. Soit j un indice de colonne de B tel que $j \leq k$. Alors, la transposée de cette colonne est soit une ligne de A, soit une ligne de Λ. S'il s'agit d'une ligne A_i de A (resp. une ligne Λ_i de Λ), alors le j-ième coefficient de w est nul (resp. est non nul) comme $A_iv = 0$ (resp. $\Lambda_iv \neq 0$). Par conséquent, le nombre d'éléments non nuls parmi les k premiers coefficients de $w = {}^tvB$ vaut c (i.e. le nombre de lignes de Λ).
3. C'est une conséquence de la propriété **IZP** et $\mathcal{N}_k({}^tvB) = c$.

□

Même si ce n'est pas commun en mathématiques, on considère dans la définition suivante des matrices à 0 ligne et m colonnes qui vont se remplir ligne par ligne par la suite.

Définition 2.7. *La tâche $t_0 =$* $\text{TASK}[\text{la matrice } 0 \times m, \text{la matrice } 0 \times m, 0, 0]$ *est appelée la* tâche initiale.

Définition 2.8. *On dit qu'une tâche $t = \text{TASK}[A, \Lambda, c, k]$ issue d'une base B de dimension $m \times n$ est une* tâche résolue *si $k = n$.*

Grâce à la proposition 2.5, une tâche résolue assure l'existence d'un vecteur v et d'un indice i tels que $\mathcal{N}({}^tvB) < \mathcal{N}(B_i)$ et $v_i \neq 0$. Cela permet de rendre plus creuse une base B, au sens du théorème 2.2.

Dans la section suivante, nous présentons les pseudo-codes des algorithmes composant CSB.

2.3.2 Algorithme CSB

Il s'agit de l'algorithme principal dont le nom signifie "Compute Sparsest Basis". Il prend en entrée une base B et renvoie une base la plus creuse B' ligne-équivalente à B. Il dépend de l'algorithme EnhanceBasis(B) qui renvoie soit une base ligne-équivalente B' telle que $\mathcal{N}(B') < \mathcal{N}(B)$ ou démontre que B était bien une base la plus creuse. Ainsi, CSB(B) itère des appels à l'algorithme EnhanceBasis(B) jusqu'à ce que la base considérée soit la plus creuse possible.

Entrée : B une base de dimension $m \times n$
Sortie : B', une base la plus creuse ligne-équivalente à B
Début
  $B' \leftarrow B$; $a \leftarrow$ **vrai** ;
  Tant que a **faire**
    $a, B' \leftarrow$ EnhanceBasis(B') ;
  Renvoyer B' ;

Algorithme 1 : CSB(B)

Arrêt. L'algorithme s'arrête car EnhanceBasis renvoie soit **faux** et B, soit **vrai** et B'. Ce dernier cas ne peut pas se produire indéfiniment car le nombre de nonzéros dans B' est positif et strictement décroissant.

Correction. L'algorithme s'arrête lorsque $a =$ **faux**, i.e. lorsque EnhanceBasis détecte que B' est une base la plus creuse.

2.3.3 Algorithme EnhanceBasis

Il dépend des algorithmes BasisToSolvedTask et EnhanceBasisUsingSolvedTask. L'algorithme BasisToSolvedTask(B) construit une tâche résolue issue de B si elle existe, ou renvoie l'ensemble nul s'il n'existe pas de telle tâche résolue. Si une telle tâche résolue peut être calculée, l'algorithme EnhanceBasisUsingSolvedTask est utilisé pour améliorer la base B.

Entrée : B une base de dimension $m \times n$
Sortie : L'un de ces deux cas : **faux** et B si B est une base la plus creuse ; **vrai** et une base B' ligne-équivalente à B telle que $\mathcal{N}(B') < \mathcal{N}(B)$ sinon
Début
  $t \leftarrow$ BasisToSolvedTask(B) ;
  Si $t \neq \emptyset$ **alors**
    $B' \leftarrow$ EnhanceBasisUsingSolvedTask(t, B) ;
    Renvoyer vrai, B' ;
  sinon
    Renvoyer faux, B ;

Algorithme 2 : EnhanceBasis(B)

Arrêt. Trivial.

Correction. L'algorithme BasisToSolvedTask renvoie soit une tâche résolue soit $\emptyset$ s'il n'existe pas de telle tâche résolue. Si $t \neq \emptyset$, l'algorithme EnhanceBasisUsingSolvedTask calcule une nouvelle base B' avec $\mathcal{N}(B') < \mathcal{N}(B)$. Sinon, puisqu'aucune tâche résolue n'existe, on a prouvé que B est la plus creuse.

2.3.4 Algorithme BasisToSolvedTask

Cet algorithme recherche un tâche résolue en explorant un arbre binaire. Il utilise une pile, que l'on initialise en la remplissant par la tâche initiale. À chaque étape de la boucle, une tâche t (où les k premières colonnes de B ont été traitées) est dépilée de la pile et deux nouvelles tâches candidates t_1 et t_2 sont construites par le traitement de la $(k+1)$-ième colonne de B. Ces tâches candidates sont empilées sur la pile si elles sont effectivement des tâches (c'est l'algorithme IsTask qui le vérifie).

Entrée : une base B
Sortie : une tâche résolue t, issue de B, ou $\emptyset$ s'il n'existe pas de telle tâche résolue (i.e. B est une base la plus creuse)

Début
  Soit S_t une pile vide ;
  Empiler la tâche initiale t_0 sur S_t ;
  Tant que $S_t \neq \emptyset$ **faire**
    Dépiler une tâche $t = \text{TASK}[A, \Lambda, c, k]$ de S_t ;
    Si $k < n$ **alors**
      // *La tâche t n'est pas résolue*
      Soit w la transposée de la $(k+1)$-ième colonne de B ;
      $A' \leftarrow \begin{pmatrix} A \\ w \end{pmatrix}$;
      $\Lambda' \leftarrow \begin{pmatrix} \Lambda \\ w \end{pmatrix}$;
      $t_1 \leftarrow [A', \Lambda, c, k+1]$; // t_1 *est peut-être une tâche*
      Si *IsTask*(t_1, B) **alors**
        Empiler t_1 sur S_t ;
      $t_2 \leftarrow [A, \Lambda', c+1, k+1]$; // t_2 *est peut-être une tâche*
      Si *IsTask*(t_2, B) **alors**
        Empiler t_2 sur S_t ;
    sinon
      Renvoyer t ;
  Renvoyer $\emptyset$;

Algorithme 3 : BasisToSolvedTask(B)

Arrêt. Soit une tâche courante t. L'algorithme s'arrête si t est résolue (i.e. si $k = n$). Sinon, t n'est pas une tâche résolue et génère de nouvelles tâches à partir de t, avec $k+1$ colonnes traitées (au lieu de k). Ce dernier cas ne peut se produire indéfiniment.

Correction. Soit une tâche non résolue t dans la boucle "tant que". On crée l'objet t_1 (resp. t_2) correspondant à l'annulation (resp. la non annulation) du $(k+1)$-ième coefficient de la combinaison linéaire des lignes de B. Les objets t_1 et t_2 sont peut-être rejetés grâce à la fonction IsTask s'ils ne sont pas des tâches (i.e. s'ils ne peuvent pas être utilisés pour diminuer le nombre de nonzéros). Par conséquent, tous les cas sont considérés et la fonction renverra $\emptyset$ si et seulement si il n'existe aucune tâche résolue issue de B.

2.3.5 Algorithme EnhanceBasisUsingSolvedTask

Cet algorithme trouve essentiellement un vecteur v et un indice i tels que $\mathcal{N}({}^tvB) < \mathcal{N}(B_i)$ et $v_i \neq 0$, qui existent nécessairement car t est une tâche résolue. Il construit ensuite une base B' plus creuse en réalisant une copie de B puis en remplaçant la ligne B'_i par ${}^tvB$.

Entrée : une tâche résolue $t = \text{TASK}[A, \Lambda, c, n]$, issue de B de dimension $m \times n$
Sortie : une base B' ligne-équivalente à B telle que B et B' ne diffèrent que d'une ligne et $\mathcal{N}(B') < \mathcal{N}(B)$

Début
  $B' \leftarrow B$;
  Calculer une base de $\text{Ker}(A)$ et la stocker par colonne dans la matrice K de dimension $m \times p$, où $p = m - \text{Rang}(A)$;
  Calculer $\mathcal{U} = \{i \in [\![1, m]\!], c < \mathcal{N}(B_i)\}$;
  Si $p = 1$ **alors**
    $v \leftarrow$ l'unique colonne de K ;
    Choisir un indice $i \in \mathcal{U}$ tel que $v_i \neq 0$;
  sinon
    $i \leftarrow 0$; $u \leftarrow 0$; // *u est le vecteur nul de dimension p*
    Tant que $i = 0$ **faire**
      $u \leftarrow \text{NextVector}(u)$;
      $v \leftarrow Ku$;
      Si $\Lambda v \not\equiv 0$ **alors**
        // *v est une solution non nulle de* $\mathcal{S}$
        Choisir un indice $i \in \mathcal{U}$ tel que $v_i \neq 0$ s'il existe ;
  $B'_i \leftarrow {}^tvB$;
  Multiplier B'_i par le PPCM des dénominateurs des éléments de B'_i ;
  Diviser B'_i par le PGCD des élements de B'_i ;
  Renvoyer B' ;

Algorithme 4 : EnhanceBasisUsingSolvedTask(t, B)

Définition 2.9. *Un sous-espace F d'un espace vectoriel E est dit* propre *si $F \neq E$.*

Lemme 2.1 (**Union finie de sous-espaces vectoriels**). *Soit $\mathbb{K}$ un corps infini. Si E est un $\mathbb{K}$-espace vectoriel, alors toute union finie de sous-espaces propres de E est strictement incluse dans E.*

Démonstration. Voir [50]. □

Théorème 2.3 (**Théorème caractéristique des solutions de $\mathcal{S}$**). *Soient les matrices $A \in \mathbb{Q}^{r\times m}$ et $\Lambda \in \mathbb{Q}^{c\times m}$. Soit $K \in \mathbb{Q}^{m\times q}$ une matrice représentant une base de* $\text{Ker}(A)$ *stockée par colonnes, et le système $\mathcal{S} : \begin{cases} Ax = 0 \\ \Lambda x \not\equiv 0 \end{cases}$.*
Les assertions suivantes sont équivalentes :

1. *$\mathcal{S}$ a des solutions non nulles dans $\mathbb{Q}^m$;*
2. $\left(\forall i \in [\![1, c]\!], \text{Rang}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix} = \text{Rang}(A) + 1\right)$ *et* $(\text{Rang}(A) \leq m - 1)$ *;*

3. $\exists u \in \mathbb{Z}^q \setminus \{0\}, \Lambda K u \not\equiv 0$.

Démonstration. (1) ⇒ (3) : On considère une solution non nulle $v \in \mathbb{Q}^m$ de $\mathcal{S}$. Donc, il existe $u' \in \mathbb{Q}^q \setminus \{0\}$ tel que $v = Ku'$ (puisque les colonnes de K forment une base de $\mathrm{Ker}(A)$). En prenant $u = \lambda u'$ (avec un entier λ bien choisi tel que $u \in \mathbb{Z}^q \setminus \{0\}$), on a $Ku = \lambda v$. Puisque λv est également une solution non nulle de $\mathcal{S}$, on a $\Lambda K u \not\equiv 0$.

(3) ⇒ (1) : On considère $v = Ku$. Alors $Av = 0$ et $\Lambda v \not\equiv 0$, donc v est une solution non nulle de $\mathcal{S}$.

(1) ⇒ (2) : On considère une solution non nulle v de $\mathcal{S}$. Puisque v est non nul et satisfait $Av = 0$, on a $\mathrm{Ker}(A) \neq \{0\}$ donc $\mathrm{Rang}(A) \leq m - 1$ (d'après le théorème du rang). Supposons qu'il existe un indice i tel que $\mathrm{Rang}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix} = \mathrm{Rang}(A)$. Alors, Λ_i est une combinaison linéaire des lignes de A donc $\Lambda_i v = 0$, donc v n'est pas une solution de $\mathcal{S}$, contradiction. On conclut donc que $\forall i \in [\![1, c]\!]$, $\mathrm{Rang}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix} = \mathrm{Rang}(A) + 1$ et $\mathrm{Rang}(A) \leq m - 1$.

(2) ⇒ (1) : L'ensemble des solutions de $\mathcal{S}$ est $V = \mathrm{Ker}(A) \setminus \bigcup_{i=1}^{c} \mathrm{Ker}(\Lambda_i) = \mathrm{Ker}(A) \setminus \bigcup_{i=1}^{c} \mathrm{Ker}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix}$. Pour tout $i \in [\![1, c]\!]$, $\mathrm{Ker}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix}$ est un sous-espace vectoriel propre de $\mathrm{Ker}(A)$ vu que $\mathrm{Rang}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix} = \mathrm{Rang}(A) + 1$. D'après le lemme 2.1, $\bigcup_{i=1}^{c} \mathrm{Ker}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix}$ est strictement inclus dans $\mathrm{Ker}(A)$. De plus, puisque $\mathrm{Rang}(A) \leq m - 1$, on a $\mathrm{Ker}(A) \neq \{0\}$, ce qui implique que $V \setminus \{0\} \neq \emptyset$. □

Arrêt. Si $p = 1$, il est trivial que l'algorithme s'arrête. Si $p \neq 1$, on doit vérifier que la boucle "tant que" s'arrête :

- comme t est une tâche résolue, il existe une solution non nulle $v \in \mathbb{Q}^m$ de $\mathcal{S}$ et un indice $j \in \mathcal{U}$ tels que $v_j \neq 0$ et $c < \mathcal{N}(B_j)$ d'après la proposition 2.5 ;
- en utilisant le troisième point du théorème 2.3, il existe un vecteur $u \in \mathbb{Z}^q$ tel que le vecteur $v = Ku$ est une solution non nulle de $\mathcal{S}$. Donc, la boucle "tant que" atteindra finalement un tel vecteur u puisque NextVector énumère tous les éléments de $\mathbb{Z}^q$.

Correction. À la ligne 2, $B' \leftarrow B$. À la fin de l'algorithme, B'_i est modifié et contient plus de zéros que B_i. En effet, $\mathcal{N}(B'_i) = c$ et puisque $i \in \mathcal{U}$, $c < \mathcal{N}(B_i)$. Au final, B' est également une base puisque $B'_i = {}^t v B$ avec $v_i \neq 0$.

2.3.6 Algorithme IsTask

Cet algorithme vérifie si une tâche candidate t est réellement une tâche, en vérifiant si t satisfait les propriétés **LCP** et **IZP**. L'objectif de cet algorithme est de détecter le plus tôt possible les tâches inutiles (i.e. les tâches qui ne nous aideront pas à rendre plus creuse notre base).

La définition suivante est nécessaire à l'algorithme IsTask et au théorème 2.5.

Définition 2.10. *Une matrice A est* ligne-unitaire d'indice j *si il existe une ligne de A avec un unique coefficient non nul à la position j.*

Proposition 2.6. *Soit M une matrice. Alors M a une colonne nulle si et seulement si* $\mathrm{Ker}(M)$ *contient au moins un vecteur avec exactement un coefficient non nul.*

Démonstration. Trivial. □

Entrée : $t = [A, \Lambda, c, k]$, satisfaisant toutes les conditions d'une tâche issue d'une base B de dimension $m \times n$, mais peut-être pas les propriétés **LCP** et **IZP**
Sortie : **vrai** si t satisfait **LCP** et **IZP** (i.e. t est une tâche), **faux** sinon

1 **Début**
2 *// **LCP** (resp. **IZP**) est vraie si les tests aux lignes 6 (resp. 11) et 3 sont faux*
3 **Si** $\left(\exists i \in [\![1, c]\!], \text{Rang}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix} = \text{Rang}(A)\right)$ *ou* $(\text{Rang}(A) = m)$ **alors**
4 **Renvoyer faux** ;
5 *// On a donc* $\left(\forall i, \text{Rang}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix} = \text{Rang}(A) + 1\right)$ *et* $(\text{Rang}(A) \leq m - 1)$
6 **Si** $(\text{Rang}(A) = m - 1)$ *et A contient au moins une colonne nulle* **alors**
7 **Renvoyer faux** ;
8 *// On a donc* $(\text{Rang}(A) \leq m - 2)$*, ou bien A n'a aucune colonne nulle*
9 Calculer A_{RREF}, la forme échelonnée réduite de A ;
10 Calculer $\mathcal{U} = \{i \in [\![1, m]\!], c < \mathcal{N}(B_i)\}$;
11 **Si** $\forall j \in \mathcal{U}$, *$A_{RREF}$ est ligne-unitaire d'indice j* **alors**
12 **Renvoyer faux** ;
13 **Renvoyer vrai** ;

Algorithme 5 : $\mathsf{IsTask}(t, B)$

Corollaire 2.3. *Soit M une matrice. Alors M n'a pas de colonne nulle si et seulement si* $\text{Ker}(M) \setminus \{0\}$ *contient uniquement des vecteurs avec au moins deux coefficients non nuls.*

Proposition 2.7. *Soient les matrices $A \in \mathbb{Q}^{r\times m}, \Lambda \in \mathbb{Q}^{c\times m}$ et le système $\mathcal{S} : \begin{cases} Ax = 0 \\ \Lambda x \not\equiv 0 \end{cases}$. Si $\mathcal{S}$ a des solutions non nulles et* $\text{Rang}(A) \leq m - 2$*, alors $\mathcal{S}$ a des solutions non nulles avec au moins deux coefficients non nuls.*

Démonstration. On suppose que $\mathcal{S}$ a des solutions et que $\text{Rang}(A) \leq m-2$. Puisque $\text{Rang}(A) \leq m-2$, on déduit du théorème du rang que $\text{Dim}(\text{Ker}(A)) \geq 2$. Par conséquent, il existe au moins deux vecteurs non nuls indépendants v_1 et v_2 solutions de $Ax = 0$. Soit v une solution non nulle de $\mathcal{S}$. Par un argument topologique, le vecteur $v + \varepsilon_1 v_1 + \varepsilon_2 v_2$ est également solution de $\mathcal{S}$ pour tous $\varepsilon_1, \varepsilon_2$ satisfaisant $|\varepsilon_1| < \varepsilon$ and $|\varepsilon_2| < \varepsilon$ (pour un petit rationnel $\varepsilon > 0$ bien choisi). On suppose que $v + \varepsilon_1 v_1 + \varepsilon_2 v_2$ a exactement un coefficient non nul pour tout $|\varepsilon_1| < \varepsilon$ et $|\varepsilon_2| < \varepsilon$. Cela implique que v_1 et v_2 ont tous les deux exactement un coefficient non nul au même indice, ce qui est impossible puisque v_1 et v_2 sont linéairement indépendants. Par conséquent, V contient au moins un vecteur non nul avec deux coefficients non nuls. □

Théorème 2.4. *Soient les matrices $A \in \mathbb{Q}^{r\times m}, \Lambda \in \mathbb{Q}^{c\times m}$ et le système $\mathcal{S} : \begin{cases} Ax = 0 \\ \Lambda x \not\equiv 0 \end{cases}$.*

Les assertions suivantes sont équivalentes :

1. *$\mathcal{S}$ a des solutions non nulles avec au moins deux coefficients non nuls ;*
2. *$\mathcal{S}$ a des solutions non nulles et au moins l'une des deux conditions suivantes est vraie :*
   - $\text{Rang}(A) \leq m - 2$,
   - *A n'a pas de colonne nulle.*

Démonstration. (1) ⇒ (2) : Puisque $\mathcal{S}$ a des solutions non nulles, alors $\text{Rang}(A) \leq m-1$. Si la condition $\text{Rang}(A) \leq m-2$ n'est pas vraie, alors on a $\text{Rang}(A) = m-1$. Par conséquent, $\text{Ker}(A)$ est généré par un vecteur contenant au moins deux coefficients non nuls. D'après le corollaire 2.3, A n'a pas de colonne nulle.

(2) ⇒ (1) : Conséquence directe du corollaire 2.3 si $\text{Rang}(A) = m-1$, ou de la proposition 2.7 si $\text{Rang}(A) \leq m-2$. □

Théorème 2.5. *Soit une matrice $A' \in \mathbb{Q}^{r\times m}$ sous forme échelonnée réduite selon les lignes. Soit K une matrice contenant une base de* $\text{Ker}(A')$ *stockée par colonnes. On suppose que* $\text{Ker}(A') \neq \{0\}$.
Pour tout indice j, les assertions suivantes sont équivalentes :

1. *A' est ligne-unitaire d'indice j ;*
2. *$K_j = 0$.*

Démonstration. (1) ⇒ (2) : Il existe une ligne $A'_i = \begin{pmatrix}0 & \cdots & 0 & 1 & 0 & \cdots & 0\end{pmatrix}$ dans laquelle le coefficient 1 est à la position j, pour $i \in [\![1, r]\!]$. Donc, toute solution v (en particulier tous les éléments de la base K) de $A'v = 0$ doit satisfaire $v_j = 0$, d'où $K_j = 0$.

(2) ⇒ (1) : À partir de $A'K = 0$, on a ${}^tK{}^tA' = 0$. Puisque $K_j = 0$, la colonne j de ${}^tK$ est nulle, ce qui implique que le vecteur canonique e_j appartient à $\text{Ker}({}^tK)$ d'après la proposition 2.6. D'où, la ligne $l = {}^te_j = \begin{pmatrix}0 & \cdots 0 & 1 & 0 & \cdots 0\end{pmatrix}$, dans laquelle le 1 est à la position j, est une combinaison linéaire des lignes de A'. Puisque A' est sous forme échelonnée réduite selon les lignes, si la combinaison l des lignes de A' fait intervenir strictement plus d'une ligne de A', l devrait au moins contenir deux coefficients non nuls (correspondant aux pivots). Donc, l est une ligne de A', et A' est ligne-unitaire d'indice j. □

Arrêt. Trivial.
Correction.

1. Si la première condition (ligne 3) est vraie, alors $\mathcal{S}$ n'a pas de solutions non nulles dans $\mathbb{Q}^m$, donc t ne satisfait pas la propriété **LCP**, et on renvoie **faux**. Sinon, on a $(\forall i \in [\![1, c]\!], \text{Rang}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix} = \text{Rang}(A) + 1)$ et $(\text{Rang}(A) \leq m-1)$ donc $\mathcal{S}$ a des solutions non nulles d'après le théorème 2.3.
2. Si la deuxième condition (ligne 6) est vraie et puisque $\mathcal{S}$ contient des solutions non nulles, le théorème 2.4 nous indique que $\mathcal{S}$ n'a pas de solutions avec au moins deux coefficients non nuls, donc t ne satisfait la propriété **LCP**, et on renvoie **faux**. Sinon, t satisfait la propriété **LCP**.
3. Si la troisième condition (ligne 11) est vraie, alors pour tout $j \in \mathcal{U}$, on a $K_j = 0$ grâce au théorème 2.5. Par conséquent, toute solution v de $\mathcal{S}$ satisfait $v_j = 0$ pour tout $j \in \mathcal{U}$. Ainsi, t ne satisfait pas la propriété **IZP** et on renvoie **faux**.

   Sinon, il existe un indice $j \in \mathcal{U}$ tel que $K_j \neq 0$ d'après le théorème 2.5, donc il existe une colonne w de K telle que $w_j \neq 0$. Il peut arriver que w_j ne soit pas solution de $\mathcal{S}$. Dans ce cas, on considère une solution non nulle u de $\mathcal{S}$ avec au moins deux coefficients non nuls. Par un argument topologique, le vecteur $\bar{u} = u + \varepsilon_1 w$ est également solution de $\mathcal{S}$ pour tout rationnel ε_1 satisfaisant $|\varepsilon_1| < \varepsilon$ (où ε est un petit rationnel). Au final, on peut choisir un ε_1 approprié pour obtenir la condition $\bar{u}_j \neq 0$, et $\mathcal{S}$ satisfait la propriété **IZP**. On renvoie alors **vrai**.

2.3.7 Algorithme NextVector

Le but de l'algorithme NextVector est d'énumérer les p-uplets de $\mathbb{Z}^p$. Il est nécessaire à l'algorithme EnhanceBasisUsingSolvedTask pour obtenir une solution v du système $\mathcal{S}$, qui est composé d'équations (i.e. $Ax = 0$) et d'inéquations (i.e. $\Lambda x \not\equiv 0$). En effet, le seul moyen que nous ayons trouvé pour obtenir des solutions de $Ax = 0$ satisfaisant également $\Lambda x \not\equiv 0$ consiste à itérer des solutions of $Ax = 0$ jusqu'à ce que $\Lambda x \not\equiv 0$ soit satisfait.

Entrée : un vecteur u de dimension p à coefficients entiers
Sortie : le vecteur qui succède à u pour un certain ordre fixé dans $\mathbb{Z}^p$

Algorithme 6 : NextVector(u)

De nombreuses variantes d'implantation sont possibles pour cet algorithme, en particulier pour l'ordre dans lequel les p-uplets sont renvoyés. Néanmoins, afin de s'assurer de la terminaison de EnhanceBasisUsingSolvedTask, l'algorithme NextVector devrait itérer tous les p-uplets de $\mathbb{Z}^p$. Cela peut être réalisé par exemple en commençant par le tuple nul, puis en énumérant les tuples de $\mathbb{Z}^p$ par norme 1 croissante (où la norme 1 d'un tuple est la somme des valeurs absolues de ses éléments), et par ordre lexicographique pour les tuples de même norme 1. Par exemple, lorsque $p = 2$, les p-uplets peuvent être énumérés de la façon suivante : $(0,0)$, $(-1,0)$, $(0,-1)$, $(0,1)$, $(1,0)$, $(-2,0)$, $(-1,-1)$, $(-1,1)$, $(0,-2)$, $(0,2)$, ...

On peut également utiliser un générateur de nombres pseudo-aléatoires, pourvu qu'il ait la propriété de générer n'importe quel p-uplet avec une probabilité non nulle (de sorte à garantir l'arrêt de l'algorithme EnhanceBasisUsingSolvedTask).

2.4 Complexité et implantation

2.4.1 Complexité

Le goulot d'étranglement de l'algorithme CSB se situe dans l'algorithme BasisToSolvedTask, qui explore un arbre binaire. De nombreuses branches sont coupées grâce à la 11-ème ligne de l'algorithme IsTask. En effet, posons $d = \max\{\mathcal{N}(B_i), i \in [\![1, m]\!]\}$. Supposons que, dans notre arbre binaire, le fils gauche (resp. le fils droit) corresponde à ajouter une équation (resp. une inéquation). Si le nombre d'inéquations c d'une quelconque tâche est supérieur ou égal à d, alors l'ensemble $\mathcal{U}$ de la 10-ème ligne dans l'algorithme IsTask est vide. Par conséquent, IsTask renvoie **faux**. Cela implique que seules les branches commençant à la racine et empruntant un fils droit au plus d fois seront explorées.

Le nombre de nœuds traités à la profondeur k est égal à $\sum_{i=0}^{d} \binom{k}{i}$. Ainsi, le nombre total de nœuds traités est borné par

$$\sum_{k=0}^{n}\sum_{i=0}^{d}\binom{k}{i} = \sum_{i=0}^{d}\sum_{k=i}^{n}\binom{k}{i} = \sum_{i=0}^{d}\binom{n+1}{i+1}.$$

On montre par des calculs simples que $\sum_{i=0}^{d}\binom{n+1}{i+1} \leq 2(n+1)^{d+1}$. Par conséquent, le nombre total de nœuds traités dans BasisToSolvedTask est en $O(\min(n^d, 2^n))$. Plus d est très petit devant n, mieux l'algorithme se comporte. Expérimentalement, nous avons pu vérifier ce dernier point.

Finalement, le nombre d'appels à l'algorithme EnhanceBasis est borné par le nombre de coefficients non nuls $\mathcal{N}(B)$ de la base initiale B (puisque le nombre de coefficients non nuls décroit au moins de 1 à chaque appel de l'algorithme EnhanceBasis, sauf pour le dernier appel).

Puisque toutes les opérations des sous-algorithmes reposent sur des opérations polynomiales, on en déduit la proposition suivante :

Proposition 2.8. *Avec les notations utilisées précédemment, l'algorithme CSB(B) est de complexité $O(\mathcal{N}(B)n^3 \min(n^d, 2^n))$.*

D'après la proposition 1.2, qui montre que le problème de calculer une base la plus creuse est NP-difficile, il n'est pas étonnant que l'algorithme CSB ait une complexité exponentielle.

2.4.2 Implantation

Nous avons choisi d'implanter les algorithmes donnés dans la section 2.3 en utilisant le logiciel de calcul formel MAPLE, qui gère nativement des entiers longs et contient de nombreuses routines d'algèbre linéaire manipulant des coefficients exacts. Sans surprise, ces algorithmes peuvent être améliorés parce que de nombreux calculs inutiles sont réalisés. Par exemple, de nombreux calculs de rangs sont réalisés dans l'algorithme IsTask.

Calcul et choix de la tâche résolue

L'algorithme BasisToSolvedTask s'arrête lorqu'il rencontre pour la première fois une tâche résolue. Cette tâche résolue peut changer si l'on empile t_2 avant t_1 dans l'algorithme BasisToSolvedTask. Ce changement n'a pas de réel impact vu qu'il accélère le temps de calcul de certains exemples, et en ralentit d'autres.

Nous avons expérimenté une autre stratégie consistant à calculer l'ensemble de toutes les tâches résolues issues de B au lieu de nous arrêter à la première tâche résolue. Dès que cet ensemble est calculé, on peut choisir la tâche résolue qui réduit le plus de nombre de coefficients non nuls de B, et conserver uniquement les tâches résolues avec $\mathrm{Rang}(A) = m - 1$ si on en rencontre une. En effet, ces tâches résolues mènent à des calculs simples dans l'algorithme EnhanceBasisUsingSolvedTask puisque $p = 1$.

Il n'est pas clair si la stratégie de calculer toutes les tâches résolues est meilleure ou moins bonne que celle qui consiste à s'arrêter sur la première tâche résolue. En effet, rechercher toutes les tâches résolues est évidemment plus coûteux, mais choisir une tâche résolue convenable pourrait diminuer le nombre d'appels ultérieurs à l'algorithme EnhanceBasis.

Utiliser la forme échelonnée réduite

Il est possible d'exiger qu'une tâche TASK$[A, \Lambda, c, k]$ satisfasse des propriétés supplémentaires. Par exemple, on peut exiger que A soit sous forme échelonnée réduite sans ligne nulle. De plus, on peut exiger que les lignes de Λ soient réduites par rapport à la matrice A dans le sens suivant : une ligne Λ_i est réduite par rapport à A si Λ_i a des coefficients nuls à chaque indice de colonne des pivots de A (réduire une ligne Λ_i par A peut être fait en soustrayant des multiples des lignes de A à Λ_i pour obtenir des zéros à ces indices de colonne des pivots de A).

Ces critères ont de nombreux avantages, en particulier dans les algorithmes 4 et 5. Dans l'algorithme 4, le calcul de $\mathrm{Rang}(A)$ est immédiat puisqu'il vaut le nombre de lignes de A. De plus, la condition $\mathrm{Rang}\begin{pmatrix} A \\ \Lambda_i \end{pmatrix} = \mathrm{Rang}(A) + 1$ peut être vérifiée immédiatement : en effet, puisque Λ_i est réduite par rapport à A, la condition est vraie si et seulement si la ligne Λ_i n'est

pas une ligne nulle. Dans l'algorithme 5, le calcul de la matrice K est immédiat et peut être réalisée en réarrangeant simplement les entrées de A en une nouvelle matrice K.

Finalement, puisque les lignes Λ_i sont réduites par rapport à A, il est facile de détecter que deux lignes Λ_i et Λ_j sont égales modulo une combinaison linéaire de lignes de A. En effet, si c'est le cas, les deux lignes sont nécessairement proportionnelles (et donc on peut éliminer l'une d'entre elles).

2.4.3 Améliorations

Les algorithmes RREF, LLL, HNF permettent parfois d'obtenir une base plus creuse. Comme ces algorithmes sont de complexités polynomiales, et donc moins coûteux que CSB, il peut s'avérer avantageux d'appliquer ces algorithmes sur la base initiale B avant d'utiliser CSB.

Ce pré-traitement est particulièrement intéressant pour les bases de grande taille, car ce pré-traitement peut diminuer le coût de calcul drastiquement, notamment pour le modèle 175 (voir section 2.5 pour plus de détails). Nous avons implanté une fonction CSBprime qui effectue ce pré-traitement avant d'appeler CSB.

Quitte à permuter des lignes et des colonnes de B, on peut également transformer B sous la forme d'une matrice diagonale par blocs où chacun des blocs est aussi une base. On peut alors calculer indépendamment les bases les plus creuses de chacun de ces blocs sans avoir besoin des lignes des autres blocs. Cela permet de calculer une base la plus creuse de la base initiale en combinant des bases les plus creuses de ses blocs diagonaux. Nous avons implanté une fonction CSBblocs qui découpe une matrice B en plusieurs sous-matrices indépendantes dont on calcule une base la plus creuse, puis qui les réassemble pour renvoyer une base la plus creuse de B. Cette fonction utilise également le pré-traitement discuté dans le paragraphe précédent.

2.5 Expérimentations

Nous avons choisi de développer notre méthode en la testant sur la base de données BioModels [1]. Nous avons pu constater que le calcul de bases les plus creuses de lois de conservation pour la plupart des modèles est résolu relativement facilement, bien que ce problème soit NP-difficile d'après la proposition 1.2.

Un premier travail a consisté à extraire des données de la base de données BioModels. Nous avons programmé un script en shell afin de télécharger les fichiers sous format SBML décrivant les modèles réarrangés. Ensuite, nous avons écrit des programmes en C utilisant la bibliothèque libSBML [7] pour extraire de ces fichiers les matrices de stœchiométrie des modèles, ainsi que les noms des espèces intervenant dans les systèmes de réactions chimiques que les matrices de stœchiométrie décrivent.

Les matrices de stœchiométrie ainsi que les espèces ont été sauvegardées dans des fichiers sous forme de matrices et de vecteurs en utilisant la syntaxe Maple.

Parmi les modèles réarrangés, nous avons sélectionné tous les modèles impliquant un seul compartiment, avec des coefficients stœchiométriques rationnels et entiers. Après avoir rejeté les modèles sans lois de conservation, nous avons considéré 229 modèles.

Après avoir calculé, avec le logiciel de calcul formel Maple, une base de lois de conservation pour chacun de ces 229 modèles, notre méthode permet de détecter que 153 modèles sont déjà les plus creux. Les calculs sur les bases des modèles BIOMD205 (de dimension 205×194) et BIOMD457 (de dimension 141×166) ont été abandonnés de l'analyse de données car leurs calculs n'étaient pas terminés après une semaine d'exécution avec la version v_2'. Dans le reste de cette section, nous considérons uniquement les 74 modèles restants.

Dans toute cette section, la version v_1 représente la version non améliorée de CSB, décrite comme à la section 2.3.2. La version v_2 représente la version par blocs de CSBblocs comme décrit dans la section 2.4.3. Les versions v_1' et v_2' représentent ces mêmes versions avec le pré-traitement de l'algorithme CSBprime, comme décrit dans la section 2.4.3. Nous présentons dans cette section les résultats de nos expérimentations sur un échantillon de ces 74 modèles pour des questions de lisibilité. Le détail des résultats de nos expérimentations est donné dans les annexes A, B et C. Nos expérimentations ont été mesurées sur un Pentium Xeon 3.40GHz avec 32Gb de mémoire.

2.5.1 Temps d'exécution des différentes versions de CSB

Modèle	m	n	$\mathcal{N}(B)$	$\mathcal{N}(B')$	$t(v_1)$	$t(v_2)$	$t(v_1')$	$t(v_2')$
BIOMD019	15	61	116	90	55.92	21.41	10.38	4.08
BIOMD086	5	17	38	32	3.60	3.67	1.38	1.32
BIOMD152	11	64	177	92	316.85	316.06	10.43	10.50
BIOMD153	11	75	162	111	1581.36	1580.29	75.36	75.58
BIOMD175	24	118	241	188	-	-	29786.17	13357.17
BIOMD186	5	11	23	18	0.70	0.59	0.14	0.16
BIOMD220	18	58	135	79	618.33	126.21	7.48	7.75
BIOMD223	20	84	130	97	86.54	15.73	2.24	0.94
BIOMD332	16	78	253	140	55386.79	32329.70	29250.66	15226.30
BIOMD333	12	54	156	97	1014.88	979.48	564.79	371.66
BIOMD334	13	73	229	108	644.73	641.46	2.40	2.57
BIOMD362	10	34	73	52	30.33	10.21	0.52	0.59
BIOMD452	23	109	203	134	371.01	375.23	12.16	8.17
BIOMD475	7	23	48	40	24.77	14.49	25.12	14.71
BIOMD478	11	33	62	47	6.71	0.88	2.18	0.59
BIOMD488	15	69	63	50	5.16	0.33	1.79	0.41
BIOMD489	20	53	70	59	13444.08	1.61	3826.15	1.65
BIOMD494	16	80	78	70	2.98	0.36	0.21	0.42
BIOMD578	8	76	120	88	8.76	8.59	1.30	1.12

FIGURE 2.1 – Temps d'exécution des différentes versions de CSB sur un échantillon de modèles avec B base de dimension $m \times n$, $\mathcal{N}(B)$ son nombre de nonzéros, $\mathcal{N}(B')$ le nombre de nonzéros d'une base la plus creuse de B, $t(v_i)$ le temps d'exécution (en s) de la version v_i de CSB, $t(v_i')$ le temps d'exécution (en s) de la version v_i' de CSB

La figure 2.1 détaille les temps d'exécution des différentes versions de CSB pour un échantillon de modèles. Les dimensions m et n de la base de lois de conservation initiale B, son nombre de coefficients non nuls et le nombre de coefficients non nuls d'une base la plus creuse B' calculée à partir de B sont également donnés dans les colonnes $\mathcal{N}(B)$ et $\mathcal{N}(B')$. Entre la version v_1 qui est la plus lente et la version v_2' qui est expérimentalement la plus rapide, on peut observer des écarts parfois très importants. En effet, on observe le plus gros écart pour le modèle BIOMD489, la version v_2' est environ 8000 fois plus rapide que la version v_1.

Pour la version v_2', 67 bases les plus creuses sur les 74 considérées sont calculées en moins de 10 secondes, 4 sont calculées entre 10 et 100 secondes, 1 est calculée en 6 minutes, 2 en environ 4 heures.

2.5.2 Gain du pré-traitement avec CSBprime

Le pré-traitement réalisé par CSBprime permet souvent d'améliorer significativement les temps d'exécution. En effet, on observe dans la figure 2.1 (et de façon plus détaillée dans l'annexe A) que les temps d'exécution de la version v_1' (resp. v_2') sont souvent plus rapides que ceux de la version v_1 (resp. v_2). Par exemple, le modèle BIOMD334 montre que la version v_1' (resp. v_2') est environ 300 fois plus rapide que la version v_1 (resp. v_2). Le modèle BIOMD475 montre que le traitement par CSBprime (pour les versions v_1 et v_2) ne permet pas toujours d'améliorer les temps d'exécution.

Modèle	m	n	$\mathcal{N}(B)$	$\mathcal{N}(B')$	$\mathcal{N}(B_{RREF})$	$\mathcal{N}(B_{LLL})$	$\mathcal{N}(B_{HNF})$
BIOMD019	15	61	116	90	121	90	121
BIOMD086	5	17	38	32	36	33	36
BIOMD152	11	64	177	92	105	101	104
BIOMD153	11	75	162	111	147	127	135
BIOMD175	24	118	241	188	200	190	200
BIOMD220	18	58	135	79	89	79	89
BIOMD223	20	84	130	97	100	97	100
BIOMD332	16	78	253	140	193	191	165
BIOMD334	13	73	229	108	118	108	118
BIOMD362	10	34	73	52	54	52	54
BIOMD452	23	109	203	134	158	135	158
BIOMD475	7	23	48	40	68	53	66
BIOMD478	11	33	62	47	51	49	51
BIOMD488	15	69	63	50	67	50	67
BIOMD489	20	53	70	59	63	63	63
BIOMD578	8	76	120	88	114	91	114

FIGURE 2.2 – Nombre de nonzéros après application des algorithmes matriciels de CSBprime
avec B base de dimension $m \times n$, $\mathcal{N}(B)$ son nombre de nonzéros, $\mathcal{N}(B')$ le nombre de nonzéros d'une base la plus creuse de B, et $\mathcal{N}(B_{RREF})$, $\mathcal{N}(B_{LLL})$, $\mathcal{N}(B_{HNF})$ les nombres de nonzéros des bases calculées à l'aide des algorithmes RREF, LLL et HNF à partir de B

La figure 2.2 (et de façon plus détaillée dans l'annexe B) montre les effets du pré-traitement par CSBprime sur le nombre de nonzéros de bases obtenues (on remplace B par une base calculée dans CSBprime avec le moins de nonzéros possible avant d'utiliser CSB). On observe que les algorithmes RREF, LLL et HNF permettent souvent d'obtenir une base avec un nombre de nonzéros plus petit que celui de B. Par exemple, le modèle BIOMD334 a une base initiale contenant 229 nonzéros et l'algorithme LLL permet de la transformer en une base ayant 108 nonzéros (dans ce cas, cela correspond au nombre de nonzéros d'une base la plus creuse issue de B). Le modèle BIOMD475 montre que les algorithmes RREF, LLL et HNF peuvent augmenter le nombre de nonzéros. Cela justifie que le temps d'exécution pour ce modèle donné dans le paragraphe précédent n'est pas amélioré.

La figure 2.3 illustre sous forme de diagramme les nombres de nonzéros observés dans la figure 2.2. Il permet de visualiser ces nombres de nonzéros plus facilement. On voit notamment, pour le modèle BIOMD475, que les algorithmes RREF, LLL et HNF ne permettent pas de diminuer le nombre de nonzéros de la base initiale. On voit également, pour le modèle BIOMD334, que l'algorithme LLL permet de trouver une base la plus creuse.

Le pré-traitement par CSBprime suffit pour 52 bases parmi nos 74 bases à calculer une base la plus creuse.

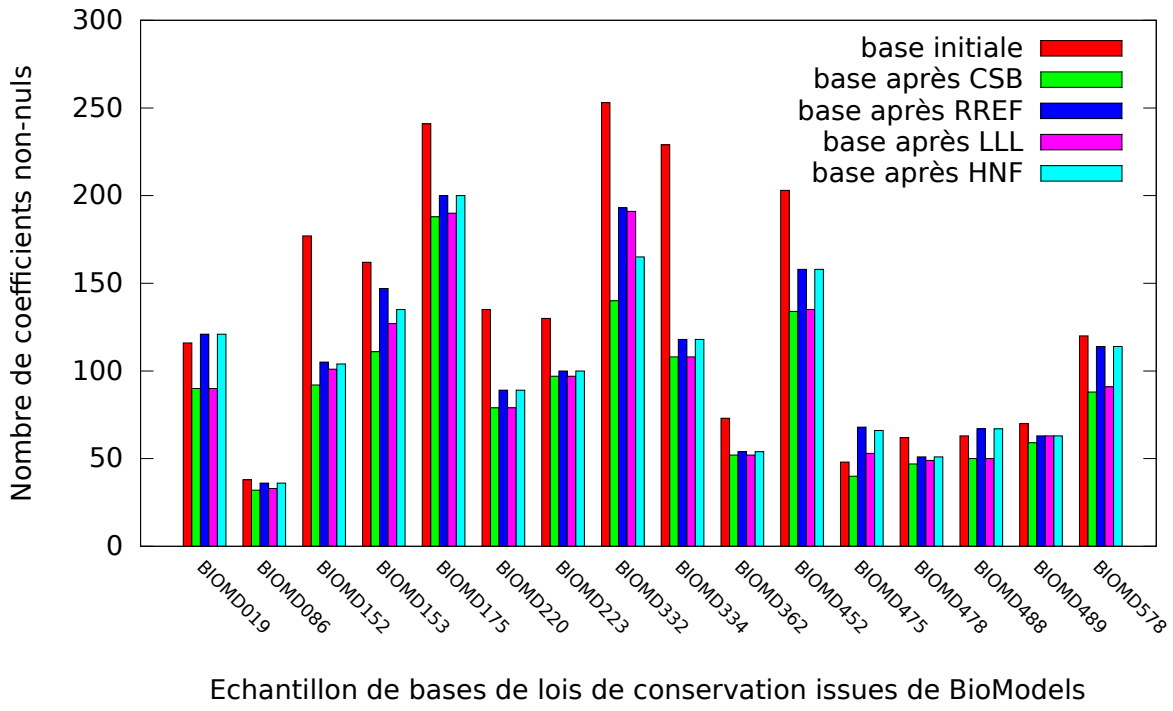

FIGURE 2.3 – Diagramme - Nombre de nonzéros après application des algorithmes matriciels de CSBprime

2.5.3 Gain du découpage par blocs et influence de certains paramètres

Le découpage par blocs avec CSBblocs permet également souvent d'améliorer significativement les temps d'exécution. En efet, on observe dans la figure 2.1 (et de façon plus détaillée dans l'annexe A) que les temps d'exécution de la version v_2 (resp. v_2') sont souvent plus rapides que ceux de la version v_1 (resp. v_1'). Par exemple, le modèle BIOMD489 montre que la version v_2 (resp. v_2') est environ 8000 fois (resp. 2000 fois) plus rapide que la version v_1 (resp. v_1'). Le modèle BIOMD334 montre que le découpage par blocs n'a pas été fructueux.

La figure 2.4 (et de façon plus détaillée dans l'annexe C) montre les valeurs des paramètres qui influencent les temps d'exécution de CSB pour la version v_2'. Nous avons vu dans la section 2.4 que la complexité dépendait du nombre $d_B = \max\{\mathcal{N}(B_i), i \in [\![1, m]\!]\}$ ainsi que des dimensions m et n de B. Le nombre d'appels à EnhanceBasis dans la boucle "tant que" de CSB, donné par $nbEtapes$, a aussi son importance sur les temps d'exécution ; il peut varier selon la façon dont les algorithmes sont codés. Enfin, nous avons vu que diviser les calculs en traitant plusieurs blocs (un nombre $nbBlocs$) peut diminuer les temps de calculs.

On observe le plus grand nombre d'étapes $nbEtapes$ dans le modèle BIOMD332, qui rend sa base de lois de conservation la plus creuse en 19 étapes. Cette base n'est découpée qu'en deux blocs et son nombre $d_B = 49$ est élevé. Le temps d'exécution $t(v_2')$ est le plus élevé. Ces 19 étapes sont principalement dues au fait que la base initiale a un nombre de nonzéros éloigné du nombre de nonzéros optimal, et que le pré-traitement via CSBprime ne suffit pas à s'en rapprocher suffisamment. Chaque appel à EnhanceBasis ne rend plus creuse qu'une loi de conservation, et il se peut qu'il faille plusieurs étapes pour que cette loi fasse effectivement partie d'une base la plus creuse recherchée.

Modèle	m	n	d_B	$d_{B'}$	$nbEtapes$	$nbBlocs$
BIOMD019	15	61	13	11	5	3
BIOMD086	5	17	12	7	2	1
BIOMD109	4	50	20	20	1	1
BIOMD152	11	64	32	23	6	1
BIOMD153	11	75	38	26	8	1
BIOMD175	24	118	42	32	9	5
BIOMD186	5	11	7	5	1	1
BIOMD220	18	58	19	8	1	1
BIOMD223	20	84	27	15	12	12
BIOMD332	16	78	49	26	19	2
BIOMD333	12	54	32	19	9	1
BIOMD334	13	73	50	16	2	1
BIOMD362	10	34	21	9	1	1
BIOMD452	23	109	28	13	11	5
BIOMD475	7	23	14	11	4	1
BIOMD478	11	33	11	10	6	3
BIOMD488	15	69	20	19	13	12
BIOMD489	20	53	11	11	4	4
BIOMD494	16	80	14	12	15	15
BIOMD578	8	76	28	28	5	1

FIGURE 2.4 – Paramètres ayant une influence sur le temps d'exécution de la version v'_2
avec B base de dimension $m \times n$, $d_B = \max\{\mathcal{N}(B_i), i \in [\![1, m]\!]\}$, $d_{B'} = \max\{\mathcal{N}(B'_i), i \in [\![1, m]\!]\}$, *nbEtapes* le nombre d'appels à EnhanceBasis, *nbBlocs* le nombre de blocs formant B (à des permutations de lignes et de colonnes près)

2.5.4 Taux de réduction

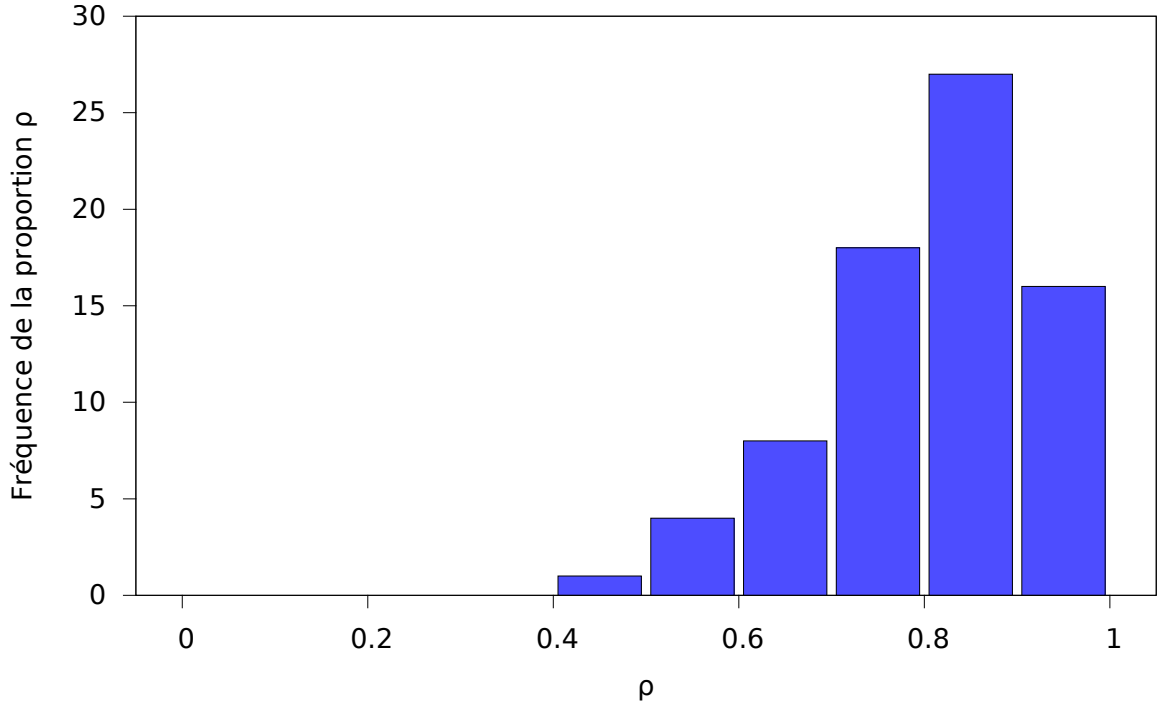

FIGURE 2.5 – Nombre de bases donnant certaines proportions $\rho = \frac{\mathcal{N}(B')}{\mathcal{N}(B)}$

Cette section a pour but de montrer à quelle proportion CSB diminue le nombre de nonzéros sur les 74 modèles considérés. Nous introduisons donc, pour toute base B, la proportion $\rho = \frac{\mathcal{N}(B')}{\mathcal{N}(B)}$ où B' est une base la plus creuse de B. Si B n'est pas la plus creuse possible, cette proportion satisfait $0 < \rho < 1$. La Figure 2.5 représente la fréquence de la proportion ρ sur des intervalles de $]0; 1[$. Elle montre que certaines bases peuvent réduire leur nombre de nonzéros au maximum de moitié.

Chapitre 3

Calcul de bases creuses par la programmation linéaire

Sommaire

3.1 Introduction 38
3.2 Mise en place d'un programme linéaire 39
3.2.1 Système non-linéaire NLP 39
3.2.2 Équivalence de NLP avec un programme linéaire LP 41
3.2.3 Sommets du polyèdre défini par LP 44
3.3 Simplexe et algorithmes 47
3.3.1 Programme linéaire sous forme canonique faisable 47
3.3.2 Algorithme InitialisationSimplexe 49
3.3.3 Algorithme CSBsimplexe 51
3.3.4 Complexité et implantation 52
3.4 Expérimentations 53
3.4.1 Liste de fonctions d'objectifs $\mathcal{L}_1$ 53
3.4.2 Liste de fonctions d'objectifs $\mathcal{L}_2$ 54
3.4.3 Pré-traitement par CSBsimplexe avec $\mathcal{L}_1$ pour CSB 55
3.4.4 Calculs sur des modèles trop volumineux pour CSB 56

Dans le chapitre 2, nous avons donné un algorithme permettant de calculer des bases les plus creuses, dont la complexité peut être exponentielle dans le pire des cas. Ce chapitre a pour but de donner une autre façon de calculer des bases creuses, avec un nouvel algorithme, basé sur de la programmation linéaire (voir [21, 22, 52]). Dans le cas de l'algorithme du simplexe, dans le pire des cas, le temps de calcul peut être exponentiel [3]. Il est cependant très efficace en pratique, et nous nous en servons pour extraire des bases creuses (mais pas forcément les plus creuses) sur la base de données BioModels.

La section 3.1 illustre un programme linéaire permettant de retrouver une base la plus creuse du modèle BIOMD361 [48] de la base de données BioModels [1, 48] manipulé à la section 2.1. Ensuite, nous discutons de la mise en place de programmes linéaires dans la section 3.2 dont les solutions optimales sont des vecteurs d'une base la plus creuse, notamment grâce à l'algorithme du simplexe. Nous présentons ensuite l'algorithme permettant de rechercher des

3. Nous savons que les programmes linéaires en variables réelles peuvent être résolus avec une complexité polynomiale en le nombre de variables (algorithme de l'ellipsoïde) mais nous n'avons pas étudié la possibilité de l'utiliser.

vecteurs creux d'une base comme le faisait l'algorithme CSB. Nous terminons ce chapitre par des expérimentations dans la section 3.4.

3.1 Introduction

Reprenons l'exemple du modèle BIOMD361 [48] de la base de données BioModels [1, 48] manipulé à la section 2.1 et introduit dans l'exemple 1.1.

Nous introduisons la famille de programmes linéaires

$$\mathcal{P}_k : \begin{cases} {}^tMx = 0 & (3.1) \\ \sum_{i=1}^{n} x_i = 1 & (3.2) \\ z = x_k[\max] & (3.3) \\ x \in \mathbb{Q}_+^n & (3.4) \end{cases}$$

indicés par un entier $k \in [\![1, n]\!]$.

La ligne (3.1) de $\mathcal{P}_k$ signifie que x est une loi de conservation. La ligne (3.4) de $\mathcal{P}_k$ signifie que l'on cherche des coefficients positifs. La ligne (3.2) de $\mathcal{P}_k$ normalise x. La ligne (3.3) de $\mathcal{P}_k$ signifie que parmi les vecteurs x solutions des lignes (3.1), (3.2) et (3.4) de $\mathcal{P}_k$, nous cherchons des vecteurs ayant la plus grande valeur possible pour la variable $z = x_k$. On dit que z est l'*objectif* et que $x \mapsto x_k$ est la *fonction d'objectif*. De façon générale, une fonction d'objectif linéaire g s'écrit sous la forme $g(x) = {}^tcx = \sum_{i=1}^{n} c_i x_i$ avec $c \in \mathbb{Q}^n$. Les vecteurs solutions des lignes (3.1), (3.2) et (3.4) de $\mathcal{P}_k$ sont appelés *solutions réalisables* (ils ne maximisent pas nécessairement z). Les vecteurs solutions des lignes (3.1), (3.2) et (3.4) de $\mathcal{P}_k$ qui maximisent l'objectif z sont appelés *solutions optimales*.

L'une des idées principales motivant ce chapitre est que, dans un programme linéaire $\mathcal{P}_k$, maximiser la variable x_k devrait minimiser les autres variables. Dans cette section, les variables sont les coefficients x_i de x (lorsque nous parlerons de ces coefficients, il faut les considérer comme des variables et non des paramètres fixés). Il est naturel d'espérer que l'on obtienne un maximum de coefficients nuls en maximisant $z = x_k$. Cela devrait permettre de trouver des vecteurs creux, comme ceux que nous avons cherchés dans le chapitre 2.

Soit B_{Pinv} une base de lois de conservation construite à partir des P-invariants calculés par Nicotine à l'exemple 1.8 :

$$B_{Pinv} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}. \tag{3.5}$$

L'ensemble des solutions réalisables de $\mathcal{P}_k$ est non vide puisque $\mathrm{Ker}({}^tM) \neq \emptyset$ et que n'importe quelle solution de $\mathrm{Ker}({}^tM)$ normalisée est une solution réalisable. Par exemple, les 3 vecteurs suivants, obtenus en normalisant les lignes de B_{Pinv}, sont des solutions réalisables de $\mathcal{P}_k$:

$$\begin{aligned} {}^tv^1 &= \begin{pmatrix} 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 0 & 0 & 1/6 \end{pmatrix}, \\ {}^tv^2 &= \begin{pmatrix} 1/4 & 0 & 0 & 1/4 & 1/4 & 0 & 1/4 & 0 \end{pmatrix}, \\ {}^tv^3 &= \begin{pmatrix} 1/3 & 0 & 0 & 0 & 0 & 1/3 & 0 & 1/3 \end{pmatrix}. \end{aligned}$$

Comme les solutions réalisables sont bornées grâce aux lignes (3.2) et (3.4) de $\mathcal{P}_k$, $\mathcal{P}_k$ admet toujours une (ou parfois plusieurs) solutions optimales finies, et ce quel que soit k.

En faisant varier k entre 1 et $n = 8$, on retrouve les vecteurs v^1, v^2, v^3. En utilisant par exemple l'agorithme du simplexe, on montre en effet que :

- v^1 est une solution optimale de $\mathcal{P}_2$ et $\mathcal{P}_3$;
- v^2 est une solution optimale de $\mathcal{P}_4$, $\mathcal{P}_5$, et $\mathcal{P}_7$;
- v^3 est une solution optimale de $\mathcal{P}_1$, $\mathcal{P}_6$, et $\mathcal{P}_8$.

Cependant, ces programmes linéaires $\mathcal{P}_k$ ne permettent pas de trouver la loi de conservation $\begin{pmatrix}0 & 1 & 1 & 0 & 0 & 0 & -1 & 1\end{pmatrix}$ qui apparaît dans la matrice B'' définie dans la section 2.1 (B'' est une base la plus creuse issue de B_{Pinv}). Ce ne sera pas un problème dans la suite de ce chapitre. En effet, nous verrons dans la section 3.2 comment mettre en place des programmes linéaires permettant de trouver des lois de conservation à coefficients négatifs. En posant $x = p - q$ avec $p, q \in \mathbb{Q}_+^n$, on considère des programmes linéaires de la forme

$$\begin{cases} {}^tM(p-q) = 0 \\ \sum_{i=1}^{n} p_i + q_i = 1 \\ z = p_k - q_k[\max] \\ p, q \in \mathbb{Q}_+^n \end{cases} \tag{3.6}$$

ainsi que des variantes de tels programmes linéaires sur les objectifs pour calculer des vecteurs d'une base la plus creuse.

3.2 Mise en place d'un programme linéaire

Soit $M \in \mathbb{Z}^{n\times q}$ une matrice de stœchiométrie (M peut également être dans $\mathbb{Q}^{n\times q}$). Les lois de conservation recherchées sont représentées par des vecteurs de $\mathrm{Ker}({}^tM)$. Soit $(b^1, \ldots, b^n)$ une base la plus creuse de $\mathrm{Ker}({}^tM)$ dont les vecteurs b^k sont dans $\mathbb{Q}^n$.

On s'intéresse dans cette section à mettre en place un programme linéaire pour trouver ces vecteurs b^k.

Comme une loi de conservation est définie à un coefficient multiplicatif près, on va normaliser avec la norme 1 les équations pour éliminer ce degré de liberté. Cette normalisation, linéaire si les variables $x_i \geq 0$, rend le domaine des solutions réalisables borné.

Tout vecteur b^k est solution de l'équation ${}^tMx = 0$ en la variable x. Il ne manque plus qu'à trouver une fonction d'objectif f, si elle existe, qui est maximisée par le vecteur b^k, afin que la programmation linéaire nous permette de trouver b^k. L'existence de cette fonction f sera justifiée par la proposition 3.11.

On introduit dans la section 3.2.1 un premier système, non-linéaire, inspiré des paragraphes précédents. Nous transformons ensuite ce système en un programme linéaire dans la section 3.2.2. Enfin, nous justifions dans la section 3.2.3 que les vecteurs formant une base la plus creuse sont des solutions optimales de programmes linéaires pour des fonctions d'objectif judicieusement choisies.

3.2.1 Système non-linéaire NLP

Soit A une matrice, qui dans notre cas est égale à ${}^tM$. Dans cette section, nous introduisons le système $\mathcal{S}_{NLP}(A, f)$, défini à la définition 3.11, dont les solutions sont potentiellement des vecteurs creux b^k (à une constante multiplicative près).

La proposition 3.10 garantit que les vecteurs b^k que nous cherchons sont effectivement solutions de systèmes du type $\mathcal{S}_{NLP}(A, f)$.

Dans tout ce chapitre, les solutions des systèmes considérés sont dans $\mathbb{Q}$. Cependant, après avoir été trouvées, ces solutions peuvent être multipliées par le PPCM des dénominateurs pour être à coefficients entiers (car les lois de conservation sont définies à un coefficient multiplicatif près).

Définition 3.11. *Soit $A \in \mathbb{Q}^{q\times n}$ une matrice. Soit $f : \mathbb{Q}^n \longrightarrow \mathbb{Q}$ une application linéaire. On définit le système*

$$\mathcal{S}_{NLP}(A,f) : \begin{cases} Ax = 0 \\ \|x\|_1 = 1 \\ z = f(x)[\max] \\ x \in \mathbb{Q}^n \end{cases} . \tag{3.7}$$

Proposition 3.9. *Soit $\mathcal{S}_{NLP}(A,f)$ défini comme dans la définition 3.11 avec* $\mathrm{Rang}(A) < n$. *Soit $X = \{x \in \mathbb{Q}^n, Ax = 0, ||x||_1 = 1\}$. Alors $X \neq \emptyset$, f est bornée sur X et atteint ses bornes sur X.*

Démonstration. D'après le théorème du Rang, $\dim(\mathrm{Ker}(A)) = n - \mathrm{Rang}(A) > 0$ donc $X \neq \emptyset$. X est clairement fermé et borné donc X est un compact. La fonction f est linéaire donc f est continue. On peut considérer $f : X \to \mathbb{Q}$. D'après [3, Corollaire 3.16], f est bornée et atteint ses bornes. □

Ainsi, si $\mathrm{Rang}(A) < n$, le système $\mathcal{S}_{NLP}(A,f)$ contient au moins une solution qui maximise f. Il admet donc toujours des solutions finies. Dans toute la suite, on considère que $\mathrm{Rang}(A) < n$ pour garantir que $\mathcal{S}_{NLP}(A,f)$ ait au moins une solution optimale d'après la proposition 3.9. La définition suivante définit l'ensemble non vide des solutions optimales de $\mathcal{S}_{NLP}(A,f)$.

Définition 3.12. *Soit $\mathcal{S}_{NLP}(A,f)$ défini comme à la définition 3.11. On définit $NLP(A,f)$, l'espace de solutions optimales de $\mathcal{S}_{NLP}(A,f)$.*

Pour un vecteur v de X (défini à la proposition 3.9), la proposition suivante justifie l'existence d'une fonction d'objectif f telle que v soit solution optimale de $NLP(A,f)$. Cela implique en particulier que tout vecteur b^k est dans $NLP(A,f)$ pour un certain f.

Proposition 3.10. *Soit $X = \{x \in \mathbb{Q}^n, Ax = 0, ||x||_1 = 1\}$. Soit v un vecteur de X. Alors, il existe une fonction d'objectif f telle que v appartienne à $NLP(A,f)$. De plus, f peut être choisie de la forme $f(x) = \sum_{i\in\mathcal{I}_+} x_i + \sum_{j\in\mathcal{I}_-} (-x_j)$ avec $\mathcal{I}_+ = \{i \in [\![1,n]\!], v_i > 0\}$ et $\mathcal{I}_- = \{i \in [\![1,n]\!], v_i < 0\}$.*

Démonstration. Soit $v \in X$. Comme $||v||_1 = 1$, v est non nul. On partitionne $[\![1,n]\!]$ avec $\mathcal{I}_+ = \{i \in [\![1,n]\!], v_i > 0\}$ et $\mathcal{I}_- = \{i \in [\![1,n]\!], v_i < 0\}$. Soit la fonction d'objectif f définie par $f(x) = \sum_{i\in\mathcal{I}_+} x_i + \sum_{j\in\mathcal{I}_-} (-x_j)$. Ainsi définie, $f(v) = 1$. Or, pour tout $x \in X$, $|f(x)| \leq \sum_{i=1}^{n} |x_i| = \|x\|_1 = 1 = f(v)$, donc v maximise f. □

Pour utiliser la programmation linéaire, il faut que le système $\mathcal{S}_{NLP}(A,f)$ soit un programme linéaire, or $\mathcal{S}_{NLP}(A,f)$ n'en est pas un à cause du fait qu'on cherche des vecteurs x dont le signe des x_i est quelconque. En effet, bien qu'elle soit linéaire sur $\mathbb{Q}^n_+$, la norme 1 n'est pas linéaire sur $\mathbb{Q}^n$. Dans la section suivante, nous transformons le système non-linéaire $\mathcal{S}_{NLP}(A,f)$ en un programme linéaire qui lui est équivalent.

3.2.2 Équivalence de NLP avec un programme linéaire LP

Dans cette section, nous transformons notre système non-linéaire $\mathcal{S}_{NLP}(A, f)$ en un programme linéaire sous la forme standard (voir définition 3.13, inspirée de [21, page 63]). Le théorème 3.6 est le résultat principal de cette section. Sous certaines conditions, il montre que les solutions du système non linéaire $\mathcal{S}_{NLP}(A, f)$ sont équivalentes aux solutions du programme linéaire $\mathcal{S}_{LP}(A, g)$ défini à la définition 3.14.

Définition 3.13 (forme standard)**.** *Un programme linéaire, en les variables x_i avec $i \in [\![1, n]\!]$, est sous* forme standard *s'il s'écrit sous la forme :*

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1, \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2, \\ \vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots, \\ a_{q,1}x_1 + a_{q,2}x_2 + \cdots + a_{q,n}x_n = b_q, \\ c_1x_1 + c_2x_2 + \cdots + c_nx_n = \zeta[\max], \\ x_1 \geq 0,\ x_2 \geq 0,\ \ldots,\ x_n \geq 0. \end{cases} \tag{3.8}$$

avec $A \in \mathbb{Q}^{q\times n}$, $b \in \mathbb{Q}^q$, $c \in \mathbb{Q}^n$ représentant la fonction d'objectif $x \mapsto {}^t\!cx$ et $\zeta \in \mathbb{Q}$ l'objectif.

Définition 3.14. *Soit $A \in \mathbb{Q}^{q\times n}$ une matrice. Soit $g : \mathbb{Q}^{2n} \longrightarrow \mathbb{Q}$ l'application linéaire définie par $g(y) = \sum_{i=1}^{n} \alpha_i(y_i - y_{n+i})$. On définit le système*

$$\mathcal{S}_{LP}(A, g) : \begin{cases} \begin{pmatrix} A & -A \end{pmatrix} y = 0 \\ \sum_{i=1}^{2n} y_i = 1 \\ \zeta = g(y)[\max] \\ y \in \mathbb{Q}_+^{2n} \end{cases}$$

et son espace de solutions optimales $LP(A, g)$.

Le programme linéaire $\mathcal{S}_{LP}(A, g)$ admet toujours des solutions réalisables car $\mathrm{Rang}(A) < n$. De plus, les solutions optimales sont finies car le vecteur y est normalisé et les y_i sont positifs (ce qui rend le domaine des solutions réalisables borné).

Ce programme linéaire est sous forme standard. En effet, toutes les inconnues y_i sont positives et les contraintes sont des égalités.

Pour introduire le théorème 3.6 donnant l'équivalence d'un système non-linéaire $\mathcal{S}_{NLP}(A, f)$ avec un programme linéaire $\mathcal{S}_{LP}(A, g)$, nous avons besoin de la définition ainsi que des deux lemmes suivants.

Soit $x \in \mathbb{Q}^n$ un vecteur. On note x^+ et x^- les vecteurs de $\mathbb{Q}^n$ définis par $x_i^+ = \begin{cases} x_i & \text{si } x_i \geq 0, \\ 0 & \text{sinon} \end{cases}$ et $x_i^- = \begin{cases} -x_i & \text{si } x_i < 0, \\ 0 & \text{sinon} \end{cases}$ de sorte qu'on puisse écrire $x = x^+ - x^-$ de façon unique. Cette transformation de x en x^+ et x^- permettra de justifier le passage de $\mathcal{S}_{NLP}(A, f)$ vers $\mathcal{S}_{LP}(A, g)$.

Définition 3.15. *Soient $x \in \mathbb{Q}^n$ et $y = \begin{pmatrix} p \\ q \end{pmatrix} \in \mathbb{Q}^{2n}$. On dit que y est le* représentant positif *de x si et seulement si $p = x^+$ et $q = x^-$.*

Lemme 3.2. *Soient $x \in \mathbb{Q}$ et $\alpha, \beta \in \mathbb{Q}_+$ tels que $x = \alpha - \beta$. Alors $|x| = \alpha + \beta$ si et seulement si $\alpha\beta = 0$.*

Démonstration. $\underline{\Leftarrow:}$ $\alpha\beta = 0$ signifie que $\alpha = 0$ ou $\beta = 0$. Si $\alpha = 0$, alors $x = -\beta$ d'où $|x| = \beta = \alpha + \beta$. Si $\beta = 0$, alors $x = \alpha$ d'où $|x| = \alpha = \alpha + \beta$.

$\underline{\Rightarrow:}$ On suppose que $\alpha\beta \neq 0$. Soient $d = \min(\alpha, \beta) \neq 0$, $\alpha' = \alpha - d$ et $\beta' = \beta - d$. Alors, $\alpha'\beta' = 0$ et $x = \alpha - \beta = (\alpha' - d) - (\beta' - d) = \alpha' - \beta'$. Comme $x = \alpha' - \beta'$ et $\alpha'\beta' = 0$, $|x| = \alpha' + \beta' = \alpha + \beta - 2d < \alpha + \beta$. Contradiction. □

Lemme 3.3. *Soit $g : \mathbb{Q}^{2n} \longrightarrow \mathbb{Q}$ l'application linéaire définie par $g(y) = \sum_{i=1}^{n} \alpha_i(y_i - y_{n+i})$ avec $\alpha_i \in \mathbb{Q}$ pour tout $i \in [\![1, n]\!]$. Soient $\zeta_{max} \in \mathbb{Q}_+$ l'objectif maximal de $\mathcal{S}_{LP}(A, g)$ et y une solution optimale de $LP(A, g)$ le réalisant. Si $\zeta_{max} \neq 0$, alors $y_i y_{n+i} = 0$ pour tout $i \in [\![1, n]\!]$.*

Démonstration. Supposons qu'il existe un indice k tel que $y_k y_{n+k} \neq 0$.

Soit $y' \in \mathbb{Q}^{2n}$ un vecteur défini par $y'_j = \begin{cases} \frac{y_j - \varepsilon}{1-2\varepsilon} \text{ si } j = k \text{ ou } j = n+k, \\ \frac{p_j}{1-2\varepsilon} \text{ sinon} \end{cases}$ avec $0 < \varepsilon < \min(\frac{1}{2}, y_k, y_{n+k})$. Ainsi, on a $\dfrac{1}{1-2\varepsilon} > 1$ et $y'_j - y'_{n+j} = \dfrac{y_j - y_{n+j}}{1-2\varepsilon}$ pour tout $j \in [\![1, n]\!]$. On vérifie alors :

1. $(A, -A)y' = \dfrac{1}{1-2\varepsilon}(A, -A)y = 0$;
2. $\displaystyle\sum_{i=1}^{2n} y'_i = \frac{1}{1-2\varepsilon}\left(\sum_{i=1}^{2n} y_i - 2\varepsilon\right) = \frac{1-2\varepsilon}{1-2\varepsilon} = 1$;
3. $g(y') = \dfrac{g(y)}{1-2\varepsilon} = \dfrac{\zeta_{max}}{1-2\varepsilon} > \zeta_{max} > 0$.

Donc y n'est pas une solution optimale, car nous avons trouvé une solution réalisable avec une valeur d'objectif plus élevée. Contradiction. Ainsi, $\forall i, y_i y_{n+i} = 0$. □

Théorème 3.6. *Soit $f : \mathbb{Q}^n \longrightarrow \mathbb{Q}$ l'application linéaire définie par $f(x) = \sum_{i=1}^{n} \alpha_i x_i$ avec $\alpha_i \in \mathbb{Q}$ pour tout $i \in [\![1, n]\!]$. Soit $g : \mathbb{Q}^{2n} \longrightarrow \mathbb{Q}$ l'application linéaire définie par $g(y) = \sum_{i=1}^{n} \alpha_i(y_i - y_{n+i})$. Soit $z_{max} \in \mathbb{Q}^+$ l'objectif maximal de $\mathcal{S}_{NLP}(A, f)$. Soit $\zeta_{max} \in \mathbb{Q}^+$ l'objectif maximal de $\mathcal{S}_{LP}(A, g)$. Soient $x \in \mathbb{Q}^n$ et $y \in \mathbb{Q}^{2n}$ deux vecteurs tels que y soit le représentant positif de x. Alors, x est une solution de $NLP(A, f)$ et $z_{max} \neq 0$ si et seulement si y est une solution optimale de $LP(A, g)$ et $\zeta_{max} \neq 0$.*

Démonstration. $\underline{\Rightarrow :}$ Soit x solution de $NLP(A, f)$ avec $z_{max} \neq 0$. Soit y le représentant positif de x.

1. On vérifie $\begin{pmatrix} A & -A \end{pmatrix} y = A(x^+ - x^-) = Ax = 0$;
2. Par définition de y, on a $y_i y_{n+i} = x_i^+ x_i^- = 0$ pour tout i. D'après le lemme 3.2, $|x_i| = x_i^+ + x_i^-$ pour tout i donc $\displaystyle\sum_{i=1}^{2n} y_i = \sum_{i=1}^{n} x_i^+ + x_i^- = \sum_{i=1}^{n} |x_i| = \|x\|_1 = 1$;

3. On a $g(y) = \sum_{i=1}^{n} \alpha_i(x_i^+ - x_i^-) = \sum_{i=1}^{n} \alpha_i x_i = f(x) \neq 0$. Or, y vérifie les équations de $\mathcal{S}_{LP}(A, g)$ d'après les deux premiers points, donc $\zeta_{max} \neq 0$;
4. Soit y' solution optimale de $LP(A, g)$. Supposons $g(y') > g(y)$. D'après le lemme 3.3, $y'_i y'_{n+i} = 0$ pour tout $i \in [\![1, n]\!]$. Soit $x' \in \mathbb{Q}^n$ un vecteur défini par $\forall i \in [\![1, n]\!], x'_i = y'_i - y'_{n+i}$. Alors $Ax' = 0$, $||x'||_1 = \sum_{i=1}^{n} y'_i + y'_{n+i} = 1$ et $f(x') = g(y') > g(y) = f(x)$. Contradiction avec x solution de $NLP(A, f)$. Ainsi, y maximimise g.

On conclut que y est une solution optimale de $LP(A, g)$.

$\underline{\Leftarrow :}$ Soit y solution de $LP(A, g)$ avec $\zeta_{max} \neq 0$. D'après le lemme 3.3, $y_i y_{n+i} = 0$ pour tout $i \in [\![1, n]\!]$. Soit $x \in \mathbb{Q}^n$ un vecteur défini par $\forall i \in [\![1, n]\!], x_i = y_i - y_{n+i}$. Alors y est le représentant positif de x.

1. On vérifie $Ax = \begin{pmatrix} A & -A \end{pmatrix} y = 0$;
2. D'après le lemme 3.2, $|x_i| = x_i^+ + x_i^-$ pour tout i donc $\sum_{i=1}^{n} |x_i| = \sum_{i=1}^{n} x_i^+ + x_i^- = \sum_{i=1}^{2n} y_i = 1$;
3. On a $f(x) = \sum_{i=1}^{n} \alpha_i x_i = \sum_{i=1}^{n} \alpha_i(x_i^+ - x_i^-) = \sum_{i=1}^{n} \alpha_i(y_i - y_{n+i}) = g(y) \neq 0$. Or, x vérifie les équations de $\mathcal{S}_{NLP}(A, f)$ d'après les deux premiers points, donc $z_{max} \neq 0$;
4. Soit x' solution de $NLP(A, f)$. Supposons $f(x') > f(x)$. Soit y' le représentant positif de x'. Alors $\begin{pmatrix} A & -A \end{pmatrix} y' = 0$, $\sum_{i=1}^{2n} y'_i = 1$, $g(y') = f(x') > f(x) = g(y)$. Contradiction avec y solution de $LP(A, g)$. Ainsi, x maximimise f.

On conclut que x est une solution de $NLP(A, f)$. □

L'exemple 3.10 illustre l'importance de la condition $\zeta_{max} \neq 0$ dans le théorème 3.6.

Exemple 3.10. *Soit* $A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \end{pmatrix}$. *Soit* $f_1 : x \to x_1$. *Comme* $\mathrm{Ker}(A)$ *est de dimension 1, on a* $NLP(A, f_1) = \{{}^t(0, 1/2, 1/2)\}$. *La solution de* $NLP(A, f_1)$ *est donc unique et* $z_{max} = 0$.

On cherche les solutions optimales de $LP(A, g_1)$ *avec la fonction d'objectif* $g_1(y) = y_1 - y_4$. *On note* $\bar{A} = \begin{pmatrix} A & -A \end{pmatrix}$. *On a :*

$$\mathrm{Ker}(\bar{A}) = \mathrm{Vect}\left\{v^1, v^2, v^3, v^4\right\} = \mathrm{Vect}\left\{ \begin{pmatrix} 1/2 \\ 0 \\ 0 \\ 1/2 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1/2 \\ 0 \\ 0 \\ 1/2 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1/2 \\ 1/2 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1/2 \\ 1/2 \end{pmatrix} \right\} \tag{3.9}$$

Pour tout $v \in \mathrm{Ker}(\bar{A})$, *on a* $g_1(v) = 0$ *donc tout vecteur* $v \in \mathrm{Ker}(\bar{A})$ *de norme* $||v||_1 = 1$ *est solution optimale de* $LP(A, g_1)$. *En particulier,* v^1 *est donc solution optimale de* $LP(A, g_1)$ *avec* $\zeta_{max} = 0$. *Or* v^1, *de la forme* $\begin{pmatrix} u \\ u \end{pmatrix}$ *(avec* $u \in \mathbb{Q}^3$*), n'est pas le représentant positif d'un vecteur à 3 composantes (car* $v_1^1 v_4^1 \neq 0$*).*

Ainsi, le théorème 3.6 permet de construire des solutions x de $NLP(A, f)$ connaissant des solutions y de $LP(A, g)$, sous condition que $\zeta_{max} \neq 0$.

Si la fonction f est combinaison linéaire des lignes de A, alors la fonction d'objectif g sera combinaison linéaire des lignes de $\begin{pmatrix} A & -A \end{pmatrix}$. On veut éviter cette configuration, puisque dès lors que l'on détecte cette dépendance linéaire, la proposition suivante conclut que l'objectif maximal est alors nul. Dans ce cas, les conditions du théorème 3.6 ne sont pas satisfaites et la fonction d'objectif g n'est pas pertinente pour déduire d'une solution optimale pour cette fonction d'objectif une solution de $\mathcal{S}_{NLP}(A, f)$.

Proposition 3.11. *Soit $f : \mathbb{Q}^n \longrightarrow \mathbb{Q}$ l'application linéaire définie par $f(x) = \sum_{i=1}^{n} \alpha_i x_i$ avec $\alpha_i \in \mathbb{Q}$ pour tout $i \in [\![1, n]\!]$. Soit $g : \mathbb{Q}^{2n} \longrightarrow \mathbb{Q}$ l'application linéaire définie par $g(y) = \sum_{i=1}^{n} \alpha_i (y_i - y_{n+i})$. Soit $\zeta_{max} \in \mathbb{Q}_+$ l'objectif maximal de $\mathcal{S}_{LP}(A, g)$. Si $\zeta_{max} = 0$, alors toute solution x de $Ax = 0$ vérifie $f(x) = 0$.*

Démonstration. On a $\zeta_{max} \geq 0$ car si une solution optimale y était telle que $\zeta_{max} < 0$, alors $-y$ serait une solution avec un meilleur objectif ($-\zeta_{max} > 0$). Contradiction, d'où $\zeta_{max} \geq 0$.

Supposons qu'il existe une solution x de $Ax = 0$ telle que $f(x) \neq 0$, et donc $x \neq 0$. Soit $x' = x/||x||_1$, alors x' est solution de $Ax = 0$ et de norme 1. Comme f est linéaire, $f(x') = \frac{f(x)}{||x||_1} \neq 0$. Soit y le représentant positif de x'. Alors $\begin{pmatrix} A & -A \end{pmatrix} y = 0$ et $\sum_{i=1}^{2n} y_i = 1$ donc y est une solution réalisable de $\mathcal{S}_{LP}(A, g)$. Comme $g(y) = f(x') \neq 0$, on en déduit que $\zeta_{max} \geq |g(y)| \neq 0$. Par contraposée, toute solution x de $Ax = 0$ vérifie $f(x) = 0$. □

3.2.3 Sommets du polyèdre défini par LP

Les vecteurs b^k que nous recherchons sont des solutions optimales de programmes linéaires $\mathcal{S}_{LP}(A, g)$ pour des fonctions linéaires g à déterminer. Cette section justifie que les représentants positifs des vecteurs b^k sont sommets d'un polyèdre associé au système $\mathcal{S}_{LP}(A, g)$. Dans la section suivante, nous utiliserons l'algorithme du simplexe, qui permettra de trouver ces sommets.

Nous rappelons ci-dessus la définition d'un sommet d'un ensemble convexe.

Définition 3.16. *Soit P un ensemble convexe. Un point $z \in P$ est appelé* sommet *de P si z n'est pas une combinaison convexe de deux autres points de P. En d'autres termes, il n'existe pas de points $x, y \in P$ et de réel λ avec $0 < \lambda < 1$ tels que $x \neq y$ et $z = \lambda x + (1 - \lambda)y$.*

Ainsi, un sommet est un point qui n'est pas strictement inclus dans un segment de P. Une définition équivalente d'un sommet serait de considérer qu'un point z est un sommet si et seulement si z n'est pas le milieu de deux autres points x et y de P.

La définition suivante est largement inspirée de [52, Section 5.3 - Polyhedra and polytopes, page 60].

Définition 3.17. *Un sous-ensemble P de $\mathbb{Q}^n$ est un* polyèdre *s'il existe une matrice D de dimension $s \times n$ et un vecteur $b' \in \mathbb{Q}^s$ tels que $P = \{x \in \mathbb{Q}^n, Dx \leq b'\}$. Autrement dit, P est un polyèdre si et seulement si P est l'intersection d'un nombre fini de demi-espaces affines.*

Nous énonçons le théorème suivant, qui est une ré-écriture de [52, Section 5.6 - Faces, facets, and vertices - Theorem 5.7]). Ce théorème décrit les sommets d'un polyèdre P par des considérations matricielles que nous manipulerons plus loin.

Théorème 3.7. *Soient $D \in \mathbb{Q}^{s_1 \times s_2}$ une matrice et $b' \in \mathbb{Q}^{s_1}$ un vecteur. Soit $P = \{x \in \mathbb{Q}^{s_2}, Dx \leq b'\}$ un polyèdre de $\mathbb{Q}^{s_2}$ et soit $z \in P$. Soit D_z la sous-matrice de D contenant uniquement les lignes d_i de D pour lesquelles $d_i z = b'_i$. Alors, z est un sommet de P si et seulement si* $\text{Rang}(D_z) = s_2$.

Ainsi z est un sommet de P si et seulement si z satisfait s_2 égalités indépendantes dans $Dx \leq b'$. Cela implique que si $s_1 < s_2$, il n'y a pas de sommets dans P. En effet, dans ce cas, il n'y a pas assez de contraintes pour définir un sommet.

Le vecteur ligne $\begin{pmatrix} 1 & \cdots & 1 \end{pmatrix}$ de longueur ℓ est noté $\mathbb{1}_\ell$ (ou simplement $\mathbb{1}$ lorsque le contexte est clair). Pour toute matrice $A \in \mathbb{Q}^{q \times n}$, on construit la matrice $A' \in \mathbb{Q}^{r \times n}$, à partir de la forme échelon réduite de A, avec l'algorithme 7.

Entrée : Une matrice $A \in \mathbb{Q}^{q \times n}$.
Sortie : Une matrice $A' \in \mathbb{Q}^{r \times n}$ de rang plein selon les lignes, avec $\text{Rang}(A) = r \leq q$.
1 **Début**
2 Calculer la matrice $A' \in \mathbb{Q}^{q \times n}$, forme échelon réduite selon les lignes de la matrice $A \in \mathbb{Q}^{q \times n}$;
3 Retirer les lignes nulles de A' ;
4 **Renvoyer** A' ;

Algorithme 7 : RREFfullRanked(A)

On souhaite résoudre le programme linéaire $\mathcal{S}_{LP}(A, g)$ à l'aide du simplexe. La proposition suivante en propose une écriture (matricielle) sous forme standard.

Proposition 3.12. *Soit $\mathcal{S}_{LP}(A, g)$ le programme linéaire écrit sous forme standard donné par la définition 3.14. Soit $A' =$ RREFfullRanked$(A) \in \mathbb{Q}^{r \times n}$ la matrice sous forme échelon réduite issue de A. Soit $b \in \mathbb{Q}^{r+1}$ le vecteur défini par ${}^t b = \begin{pmatrix} 0 & \cdots & 0 & 1 \end{pmatrix}$. Soit $C = \begin{pmatrix} A' & -A' \\ \mathbb{1}_n & \mathbb{1}_n \end{pmatrix}$. Alors on peut écrire la forme standard suivante :*

$$\mathcal{S}_{LP}(A, g) : \begin{cases} Cy = b \\ \zeta = g(y)[\max] \\ y \in \mathbb{Q}_+^{2n} \end{cases} .$$

Démonstration. Trivial. □

On suppose dans la suite de ce chapitre que $A \in \mathbb{Q}^{r \times n}$ est de rang plein selon lignes i.e. on a déjà éliminé des lignes de A, à l'aide de RREFfullRanked (en considérant $A = A'$) par exemple. Ainsi, $\text{Rang}(A) = r$.

Corollaire 3.4. *Avec les notations de la proposition 3.12, la matrice C est de rang plein selon les lignes, i.e.* $\text{Rang}(C) = r + 1$.

Démonstration. Puisque A' est sous forme échelon réduite, la matrice $\begin{pmatrix} A' & -A' \end{pmatrix}$ l'est également. Supposons que C ne soit pas de rang plein. Alors, $\begin{pmatrix} \mathbb{1}_n & \mathbb{1}_n \end{pmatrix} = \sum_{i=1}^{r} \alpha_i \begin{pmatrix} A'_i & -A'_i \end{pmatrix}$. En découpant par blocs, on a $\mathbb{1}_n = \sum_{i=1}^{r} \alpha_i A'_i$ et $\mathbb{1}_n = -\sum_{i=1}^{r} \alpha_i A'_i$. En additionnant ces deux égalités, on obtient $2 \times \mathbb{1}_n = 0$. Contradiction. Donc C est de rang plein selon les lignes. □

On reprend la notation $\mathcal{N}(x)$ donnant le nombre de coefficients non-nuls d'un vecteur x.

Lemme 3.4. *Soient $x \in \mathbb{Q}^n$ et $y \in \mathbb{Q}^{2n}$. Si y est le représentant positif de x, alors $\mathcal{N}(x) = \mathcal{N}(y)$.*

Démonstration. Trivial. □

Soit P l'ensemble des solutions réalisables de $\mathcal{S}_{LP}(A, g)$. Le lemme suivant permet de justifier que P définit bien un polyèdre, que l'on écrit aussi sous forme d'intersection de demi-espaces.

Lemme 3.5. *Avec les notations de la proposition 3.12, l'ensemble $P = \{y \in \mathbb{Q}^{2n}_+,\ Cy = b\}$ est un polyèdre que l'on peut écrire sous la forme $P = \{y \in \mathbb{Q}^{2n},\ Dy \leq b'\}$.*

Démonstration. Comme $Cy = b$ peut être ré-écrit par $Cy \leq b$ et $Cy \geq b$, et comme $y \in \mathbb{Q}^{2n}_+$ peut être ré-écrit $I_{2n}y \geq 0$, on peut ré-écrire P sous la forme $P = \{y \in \mathbb{Q}^{2n},\ Dy \leq b'\}$ avec $D = \begin{pmatrix} C \\ -C \\ -I_{2n} \end{pmatrix}$ une matrice de $\mathbb{Q}^{2n+2r+2}$ et $b' = \begin{pmatrix} b \\ -b \\ 0 \end{pmatrix}$ un vecteur de $\mathbb{Q}^{2r+3}$. □

Soient x une solution de $NLP(A, f)$ et y son représentant positif. Le théorème suivant fait le lien entre le nombre de nonzéros de x et le fait que y soit un sommet de P.

Théorème 3.8. *On reprend les notations de la proposition 3.12. Soit $x \in NLP(A, f)$ avec $A \in \mathbb{Q}^{r\times n}$. Soit y le représentant positif de x. Soit $P = \{y \in \mathbb{Q}^{2n}_+,\ Cy = b\}$ le polyèdre représentant l'espace de solutions $LP(A, g)$. Alors, $\mathcal{N}(x) \leq r + 1$ si et seulement si y est un sommet de P.*

Démonstration. Dans cette démonstration, on note $\mathcal{Z}(z)$ le nombre de coefficients nuls du vecteur z. On a ainsi $\mathcal{N}(z) + \mathcal{Z}(z) = 2n$.

$\underline{\Rightarrow :}$ On reprend les notations de la démonstration du lemme 3.5. Comme D contient $-I_{2n}$ et que D a $2n$ colonnes, il est évident que $\text{Rang}(D) = 2n$. Pour tout vecteur z de P, on considère D_z la sous-matrice de D contenant uniquement les lignes d_i de D pour lesquelles $d_i z = b'_i$. Comme $Cz = b$, alors D_z contient nécessairement C. D'après le corollaire 3.4, $\text{Rang}(C) = r + 1$.

Soit $x \in NLP(A, f)$ tel que $\mathcal{N}(x) \leq r + 1$. Soit y le représentant positif de x, alors y est une solution optimale de $LP(A, g)$ d'après le théorème 3.6. D'après le lemme 3.4, on a $\mathcal{N}(y) = \mathcal{N}(x) \leq r + 1$. On suppose que y n'est pas un sommet de P. Alors, une conséquence directe du théorème 3.7 est que $\text{Rang}(D_y) < 2n$. On sait que D_y contient nécessairement C mais il contient également des lignes de I_{2n}. Parmi elles, on considère celles qui forment une matrice T de sorte que $\text{Rang}(D_y) = \text{Rang}(C) + \text{Rang}(T)$, i.e. T contient les lignes de $-I_{2n}$ indépendantes de C.

On a donc $\text{Rang}(C) + \text{Rang}(T) < 2n$. Or d'après le corollaire 3.4, on a $\text{Rang}(C) = r + 1$, d'où $\text{Rang}(T) \leq 2n - r - 2$. Comme T représente des équations $y_i = 0$, on a $\mathcal{Z}(y) = \text{Rang}(T)$ d'où $\mathcal{N}(y) \geq r + 2$. Contradiction, donc y est un sommet de P.

$\underline{\Leftarrow :}$ Soit y un sommet de P. Comme y satisfait $Cy = b$, on conclut grâce au théorème 3.7 que D_y contient les $r + 1$ équations données par C et donc $n - r - 1$ équations $y_i = 0$ indépendantes de C. Donc $\mathcal{Z}(y) \geq n - r - 1$, d'où $\mathcal{N}(y) \leq r + 1$. Le lemme 3.4 conclut alors que $\mathcal{N}(x) = \mathcal{N}(y) \leq r + 1$. □

Le lemme suivant montre que les vecteurs lignes d'une base la plus creuse B de $\text{Ker}(A)$ vérifient la condition $\mathcal{N}(x) \leq r + 1$ (pour $b^k = {}^t x$) du théorème 3.8.

Lemme 3.6. *Si $B \in \mathbb{Q}^{(n-r)\times n}$ est une base la plus creuse, alors $\forall i \in [\![1, n - r]\!], \mathcal{N}(B_i) \leq r + 1$.*

Démonstration. Si B est une base la plus creuse, alors elle est plus creuse que sa forme échelon réduite B'. Quitte à réordonner B' selon les colonnes, on peut ré-écrire B' sous la forme $(I_{n-r}\ E)$ I est la matrice identité et E une matrice. Alors, pour tout $i \in [\![1, n-r]\!]$, on a $\mathcal{N}(B'_i) \leq r+1$. Quitte à ré-ordonner B_i et B'_i selon leurs nombres de nonzéros croissants, la proposition 2.4 conclut que pour tout $i \in [\![1, n-r]\!]$, on a $\mathcal{N}(B_i) \leq \mathcal{N}(B'_i) \leq r+1$. □

Théorème 3.9. *Soit $x \in NLP(A, f)$ avec $A \in \mathbb{Q}^{r\times n}$ matrice de rang plein selon les lignes. Soit y le représentant positif de x. Soit $P = \{y \in \mathbb{Q}_+^{2n},\ Cy = b\}$ le polyèdre représentant l'espace de solutions $LP(A, g)$. Alors, x est un vecteur d'une base la plus creuse de A si et seulement si y est un sommet de P.*

Démonstration. L'existence d'une fonction d'objectif maximisant tout vecteur x de norme 1 est justifiée par la proposition 3.10. D'après le lemme 3.6, tout vecteur x d'une base la plus creuse est tel que $\mathcal{N}(x) \leq r+1$. Le théorème 3.8 permet de conclure cette démonstration. □

Le théorème 3.9 permet donc de trouver les vecteurs formant toute base la plus creuse du noyau d'une matrice A en résolvant $LP(A, g)$ pour une certaine fonction d'objectif g. Mais savoir quelle fonction g utiliser pour trouver tous les vecteurs d'une telle base est difficile.

La section suivante s'attache à exposer la recherche de bases les plus creuses en testant des fonctions d'objectifs qui paraissent simples.

3.3 Simplexe et algorithmes

Le théorème 3.9 montre que les solutions de $\mathcal{S}_{LP}(A, g)$ sont des sommets du polyèdre des solutions réalisables $P = \{x \in \mathbb{Q}_+^{2n},\ Cx = b\}$ avec C et b définis comme à la proposition 3.12. Ce théorème justifie l'usage de l'algorithme du simplexe, qui permet de résoudre des programmes linéaires en se déplaçant d'un sommet du polyèdre P vers un autre, tant qu'il est capable d'augmenter la valeur de l'objectif (dans le cas d'une maximisation).

L'une des difficultés de l'algorithme du simplexe est de connaître un premier sommet (c'est ce qu'on appelle le *problème de démarrage*). En effet, il se peut qu'il n'existe aucun sommet de départ et donc que le système n'ait pas de solutions réalisables. On montre dans la section 3.3.2 que l'on peut facilement calculer un tel sommet.

3.3.1 Programme linéaire sous forme canonique faisable

Les définitions suivantes sont inspirées de [21, Section 3.2 - The Simplex Algorithm] et adaptées à nos notations ainsi qu'à notre discours.

Définition 3.18. *Soit un programme linéaire de $q+1$ équations linéaires en n variables y_j (dont l'équation linéaire représentant l'objectif $\zeta = {}^t cy$), écrit sous forme standard comme dans la définition 3.13. Ce programme linéaire est dit* sous forme canonique *par rapport à un ensemble ordonné de variables $(y_{j_1}, y_{j_2}, \ldots, y_{j_q})$ si et seulement si les conditions suivantes sont vérifiées :*

- *y_{j_i} a un coefficient unitaire dans l'équation i et un coefficient nul dans les autres équations ;*
- *$\forall i \in [\![1, q]\!]$, $c_{j_i} = 0$.*

Remarque 3.4. *On reprend le système* (3.8) *de la définition 3.13. Transformer ce système sous forme canonique ressemble à une mise sous forme échelon réduite. Ce n'est cependant pas le cas puisque les q pivots choisis dans la définition 3.18 ne sont pas nécessairement les premiers coefficients non nuls d'une ligne et on ne choisit aucun pivot pour la ligne $\zeta = {}^t cy$. Il faut également que les coûts c_j soient nuls dans les colonnes j où se trouvent les pivots pour que*

le système soit sous forme canonique. Avec les notations que nous avons choisies, nous avons besoin de q pivots pour les $q+1$ équations considérées. Dans [21, page 67], il y a $q+1$ pivots pour les $q+1$ équations considérées. Avec nos notations, ce pivot manquant est en fait masqué par l'absence d'une variable fictive ζ_0 dans nos notations. Cette variable ζ_0 n'étant pas utile dans cette thèse, nous ne l'utilisons pas. Voir [21, page 67] pour plus de détails.

On rappelle que le programme linéaire $\mathcal{S}_{LP}(A, g)$ contient l'équation matricielle $Cy = b$, avec C et b définis comme à la proposition 3.12. Soit $C' = QC$ la forme échelon réduite de C avec Q inversible. Soit $\bar{b} = Qb$, on a alors $C'y = \bar{b}$. En considérant l'équation matricielle $C'y = \bar{b}$, représentant le programme linéaire $\mathcal{S}_{LP}(A, g)$ avec $r+1$ équations en n variables, la condition "y_{j_i} a un coefficient unitaire dans l'équation i et un coefficient nul dans les autres équations" de la définition 3.18 se traduit par la colonne j_i de C' contient un unique coefficient non nul (sur la ligne i) et ce coefficient vaut 1, vu que C' est sous forme échelon réduite. Soit $\mathcal{J}_B = \{j_1, j_2, \ldots, j_{r+1}\}$ l'ensemble des indices de colonnes de C' où un unique coefficient est non nul et vaut 1. Soit $\mathcal{J}_N = [\![1, n]\!] \setminus \mathcal{J}_B$. L'ensemble $\mathcal{J}_B$ (respectivement $\mathcal{J}_N$) désigne l'ensemble des variables de base (respectivement hors-base). Les colonnes de C' aux indices $j_i \in \mathcal{J}_B$ forment la matrice I_{r+1}. On note $\bar{C}$ la matrice formée par les colonnes de C' aux indices $j_i \in \mathcal{J}_N$. En posant $y_B = {}^t(y_{j_1}, y_{j_2}, \ldots, y_{j_{r+1}})$ et $y_N = {}^t(y_{j_{r+2}}, \ldots, y_{j_n})$, on transforme le système $C'y = \bar{b}$ en le système

$$I_{r+1} y_B + \bar{C} y_N = \bar{b}. \tag{3.10}$$

Les variables dans y_B sont appelés *variables de base* et celles de y_N sont appelées *variables hors-base*. Elles se déduisent respectivement de $\mathcal{J}_B$ et de $\mathcal{J}_N$.

En considérant la fonction d'objectif $g(y) = {}^t c y$, la condition "$c_{j_i} = 0$ pour tout j_i" (dans $\mathcal{J}_B$) implique que pour tout $j_i \in \mathcal{J}_B$, si $c_{j_i} \neq 0$, on réduit ${}^t c$ par rapport aux pivots de T par la combinaison linéaire ${}^t c \leftarrow {}^t c - c_{j_i} T_i$. Soit w le vecteur ligne obtenu après ces combinaisons linéaires, alors "$w_{j_i} = 0$ pour tout j_i". En posant $\bar{\zeta} = wy$, on écrit l'objectif $\bar{\zeta}$ sous la forme

$$\bar{\zeta} = 0 y_B + \bar{w} y_N [\max]. \tag{3.11}$$

Le programme linéaire (défini par les équations du système 3.10 et par l'objectif $\bar{\zeta}$ de l'équation 3.11)

$$\begin{cases} I_{r+1} y_B + \bar{C} y_N = \bar{b}, \\ \bar{\zeta} = 0 y_B + \bar{w} y_N [\max], \\ y \in \mathbb{Q}_+^{2n} \end{cases} \tag{3.12}$$

est sous forme canonique. Il est une réécriture de $\mathcal{S}_{LP}(A, g)$.

Définition 3.19. *La solution particulière obtenue en résolvant le programme linéaire* (3.12) *avec l'objectif 3.11 et en fixant les variables hors-base à zéro, i.e. $y_N = 0$, est appelée* solution de base.

Définition 3.20. *Soit $\bar{y}$ la solution de base du système* (3.12). *Si $\bar{y} = \bar{b} \geq 0$* [4]*, alors le programme linéaire est dit* sous forme canonique faisable *et $\bar{y}$ est dit* faisable.

Pour utiliser l'algorithme du simplexe sur un système qui est sous forme standard, il faut le convertir en système sous forme canonique puis en système sous forme canonique faisable. Il s'agit en fait de démarrer l'algorithme du simplexe sur une solution faisable, dont on ne dispose pas toujours. Dans la section suivante, nous verrons que l'algorithme InitialisationSimplexe permet de transformer le système $\mathcal{S}_{LP}(A, g)$ en le système (3.12) avec $T = \begin{pmatrix} \bar{C} & \bar{b} \end{pmatrix}$, où $\bar{b} \geq 0$. L'inégalité $\bar{b} \geq 0$ impliquera que le système ainsi obtenu est sous forme canonique faisable.

4. Pour un vecteur v de dimension n, la notation $v \geq 0$ signifie que pour tout i, $v_i \geq 0$.

3.3.2 Algorithme InitialisationSimplexe

Entrée : Un programme linéaire $\mathcal{S}_{LP}(A, g)$ avec A une matrice de $\mathbb{Q}^{q\times n}$ de rang r et $g(y)$ une application linéaire pouvant s'écrire $g(y) = {}^t v y$ avec v un vecteur de $\mathbb{Q}^{2n}$.
Sortie : Une matrice $T \in \mathbb{Q}^{(r+1)\times(2n+1)}$, $\mathcal{J}_B$ un ensemble d'indices de variables de base et w un vecteur ligne de $\mathbb{Q}^{2n+1}$ représentant la fonction d'objectif sur ses $2n$ premiers coefficients et $-\zeta_0$ sur son dernier coefficient.

1 **Début**
2 Soit A' la matrice de $\mathbb{Q}^{r\times n}$ définie par RREFfullRanked(A) ;
3 Soit $T \in \mathbb{Q}^{(r+1)\times(2n+1)}$ la matrice définie par $T = \begin{pmatrix} A' & -A' & 0 \\ \mathbb{1}_n & \mathbb{1}_n & 1 \end{pmatrix}$;
4 $T_{r+1} \leftarrow T_{r+1} - \sum_{k=1}^{r} T_k$;
5 $T_{r+1} \leftarrow T_{r+1}/2$;
6 $T_1 \leftarrow T_1 + T_{r+1}$;
7 Soit $\mathcal{J}_B = \{j_1, j_2, \ldots, j_r, n + j_1\}$ l'ensemble des indices de colonnes des pivots de T ;
8 Soit w le vecteur ligne de $\mathbb{Q}^{2n+1}$ défini par $w = \begin{pmatrix} {}^t v & -\zeta_0 \end{pmatrix}$; // *avec* $\zeta_0 = 0$
9 Réduire w par rapport aux pivots de T ; // $\forall j_i \in \mathcal{J}_B$, $w_{j_i} \neq 0$, *faire* $w = w - w_{j_i} T_i$
10 **Renvoyer** $T, \mathcal{J}_B, w$;

Algorithme 8 : InitialisationSimplexe$(\mathcal{S}_{LP}(A, g))$

L'algorithme InitialisationSimplexe permet d'initialiser l'algorithme du simplexe en se plaçant sur un sommet du polyèdre défini par $P = \{x \in \mathbb{Q}_+^{2n},\ Cx = b\}$. Pour définir les variables de base, il faut faire apparaître toutes les colonnes de la matrice identité I_{r+1} dans C. Les r premières lignes de la matrice A' calculée à la ligne 2 font apparaître r pivots qui seront les mêmes que T (construit à partir de A') après la combinaison linéaire de la ligne 4. Ainsi les r premières colonnes de I_{r+1} sont dans les $2n$ premières colonnes de T (et donc dans C). Les lignes 5 et 6 font apparaître un $(r+1)$-ième pivot pour T à la colonne d'indice $n + j_1$. La preuve de l'algorithme justifie tout cela plus en détail.

L'exemple ci-dessous illustre le fonctionnement de l'algorithme InitialisationSimplexe.

Exemple 3.11. *On considère la matrice de stœchiométrie du modèle BIOMD361 donnée par la figure 1.2. On pose* $A = {}^t M$ *et on s'intéresse à résoudre le programme linéaire* $\mathcal{S}_{LP}(A, g)$ *avec* $g(y) = y_2 - y_{10}$ *que l'on représentera sous la forme du vecteur ligne*

$$\left(\begin{array}{cccccccc|cccccccc} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

Soit A' *la matrice obtenue avec l'algorithme RREFfullRanked. On a*

$$A' = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -1 & -1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \end{pmatrix}.$$

On construit T *comme défini à la ligne 3. Les pivots de* T *que nous utilisons sont mis en gras.*

Suite aux traitements des lignes 4 et 5 qui modifient T_{r+1}, T s'écrit :

$$T = \left(\begin{array}{cccccccc|cccccccc|c} \mathbf{1} & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 1 & -1 & -1 & 0 & 0 & 0 & -1 & 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 1 & -1 & -1 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{3}{2} & 2 & 3 & 1 & 1 & 1 & 1 & 1 & \frac{5}{2} & -1 & -2 & \frac{1}{2} \end{array}\right).$$

Le traitement de la ligne 7 transforme T en

$$T = \left(\begin{array}{cccccccc|cccccccc|c} \mathbf{1} & 0 & 0 & 0 & 0 & -\frac{3}{2} & 1 & 2 & 0 & 1 & 1 & 1 & 1 & \frac{5}{2} & 0 & -1 & \frac{1}{2} \\ 0 & \mathbf{1} & 0 & 0 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 1 & -1 & -1 & 0 & 0 & 0 & -1 & 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 1 & -1 & -1 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{3}{2} & 2 & 3 & \mathbf{1} & 1 & 1 & 1 & 1 & \frac{5}{2} & -1 & -2 & \frac{1}{2} \end{array}\right).$$

On détermine $\mathcal{J}_B = \{1, 2, 3, 4, 5, 9\}$ à la ligne 7. Avec g, on construit le vecteur ligne w à la ligne 8 :

$$w = \left(\begin{array}{cccccccc|cccccccc|c} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

puis on réduit w par rapport aux pivots de T à la ligne 9, on obtient alors

$$w = \left(\begin{array}{cccccccc|cccccccc|c} 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \end{array}\right).$$

Preuve – InitialisationSimplexe est correct. On reprend la forme standard définie par la proposition 3.12. Soit $T = \begin{pmatrix} C & b \end{pmatrix}$ la matrice définie à la ligne 3. Par définition de b, $b \geq 0$ (son unique coefficient non nul est $b_{r+1} = 1$). Soient $j_1, j_2, \ldots, j_r$ les indices de colonnes des pivots de A'. Ces indices sont également les indices de colonnes des r premiers pivots de T. On va maintenant faire apparaître I_{r+1} dans C. Cela consiste à trouver une matrice inversible Q telle que $C' = QC$. On transformera de même b en $\bar{b}$ ($Cy = b$ devient $C'y = \bar{b}$), ce qui justifie l'usage de $T = \begin{pmatrix} C & b \end{pmatrix}$ (que l'on multipliera par Q). Dans les paragraphes qui suivent, nous ne ferons pas référence à Q mais à des combinaisons linéaires, que Q permet de faire (on peut bien entendu reconstituer la matrice Q à l'aide de ces combinaisons linéaires).

Les coefficients de la dernière ligne de T étant tous unitaires, mettre à zéro les coefficient T_{r+1,j_i} fera apparaître les r première colonnes de I_{r+1} dans T. La combinaison linéaire de la ligne 4 fait cela. En effet, pour tout j_i (avec $i \in [\![1, r]\!]$) on a $T_{r+1,j_i} - \sum_{k=1}^{r} T_{k,j_i} = 1 - A'_{i,j_i} = 1 - 1 = 0$ d'où $T_{r+1,j_i} \leftarrow 0$. On note que comme $T_{r+1,2n+1} \leftarrow T_{r+1,2n+1} - \sum_{k=1}^{r} T_{k,2n+1} = 1 - \sum_{k=1}^{r} b_k = 1 - 0 = 1$, b n'est pas modifié par la combinaison linéaire de la ligne 4.

Faire apparaître la dernière colonne de I_{r+1} dans C revient à choisir un pivot $T_{r+1,j}$ avec $j \in [\![1, n]\!]$ et effectuer les combinaisons linéaires $T_i \leftarrow T_i - T_{i,j}T_{r+1}$ pour $i \in [\![1, r]\!]$, annulant les coefficients $T_{i,j}$. Choisir $j = n + j_i$ avec $i \in [\![1, r]\!]$ fixé convient. En effet, on montre pour tout j_i que le traitement de la ligne 4 implique que $T_{r+1,n+j_i} - \sum_{k=1}^{r} T_{k,n+j_i} = 1 + A'_{i,j_i} = 1 + 1 = 2$. Nous avons choisi comme pivot le coefficient $T_{r+1,n+j_1}$, que l'on met à 1 avec l'opération de la ligne 5. Il faut maintenant annuler les coefficients $T_{i,n+j_1}$ pour tout $i \in [\![1, r]\!]$, or seul $T_{1,n+j_1} \neq 0$ (et vaut -1) comme la $(n + j_1)$-ième colonne de T contient l'opposé de la j_1-ième colonne de A' par construction de T. Le traitement de la ligne 6 finit de faire apparaître la dernière colonne de I_{r+1}. Les opérations des lignes 5 et 6 transforment b en $\bar{b}$ défini par $\bar{b}_1 = \frac{1}{2}$, $\bar{b}_{r+1} = \frac{1}{2}$ et $\bar{b}_i = 0$ pour tout $i \in [\![2, r]\!]$ donc $\bar{b} \geq 0$.

Soit $\mathcal{J}_N = [\![1, 2n]\!] \setminus \mathcal{J}_B$ l'ensemble des variables hors-base. La matrice $\bar{C}$ se compose des colonnes de T dont les indices sont dans $\mathcal{J}_N$. Nous avons donc transformé le système $\mathcal{S}_{LP}(A, g)$ sous forme canonique. Comme $\bar{b} \geq 0$, il est également sous forme canonique faisable.

Comme nous avons fait apparaître I_{r+1} dans C, nous pouvons définir l'ensemble des variables de base $\mathcal{J}_B$ comme à la ligne 7. Nous représentons enfin la fonction d'objectif ainsi que la valeur d'objectif maximal dans le vecteur w, que nous réduisons à l'aide des lignes de T, de sorte que pour tout j_i, $w_{j_i} = 0$.

□

3.3.3 Algorithme CSBsimplexe

Nous avons vu dans la section précédente que l'algorithme InitialisationSimplexe résout le problème de démarrage. Nous présentons dans cette section l'algorithme CSBsimplexe qui effectue plusieurs appels à l'algorithme du simplexe en faisant varier les fonctions d'objectifs fournies dans une liste donnée en paramètre.

Le concept de matrice à 0 ligne[5] dont nous avons parlé dans la section 2.3.1 est réutilisé dans l'algorithme CSBsimplexe.

Entrée : Une matrice A de $\mathbb{Q}^{q\times n}$ et une liste $\mathcal{L}$ de fonctions linéaires de $\mathbb{Q}^{2n}$ dans $\mathbb{Q}$.
Sortie : Une matrice $B \in \mathbb{Q}^{p\times n}$ de rang plein selon les lignes telle que les lignes de $\begin{pmatrix} B & -B \end{pmatrix}$ sont des solutions optimales de $\mathcal{S}_{LP}(A, g)$ pour des fonctions linéaires $g : \mathbb{Q}^{2n} \to \mathbb{Q}$ de $\mathcal{L}$.

1 **Début**
2 Soit $\mathcal{V}$ une liste vide ;
3 **Pour tout** g **dans** $\mathcal{L}$ **faire**
4 $T, \mathcal{J}_B, w = \text{InitialisationSimplexe}(\mathcal{S}_{LP}(A, g))$ avec $T = \begin{pmatrix} \bar{C} & \bar{b} \end{pmatrix}$ défini à la section 3.3.1 et $w = \begin{pmatrix} {}^t v & -\zeta_0 \end{pmatrix}$;
5 Soient y le sommet donné par l'algorithme du simplexe à partir de $T, \mathcal{J}_B, w$ et ζ_{max} l'objectif maximal pour y ;
6 **Si** $\zeta_{max} \neq 0$ **alors**
7 Soit x le vecteur ligne tel que y est le représentant positif de x ;
8 Ajouter x à $\mathcal{V}$;
9 Trier les élements de $\mathcal{V}$ par nombre croissant de nonzéros ;
10 Soit B la matrice de dimension $0 \times n$;
11 **Pour tout** x **dans** $\mathcal{V}$ **faire**
12 **Si** $\text{Rang}\begin{pmatrix} B \\ x \end{pmatrix} = \text{Rang}(B) + 1$ **alors**
13 $B = \begin{pmatrix} B \\ x \end{pmatrix}$;
14 **Renvoyer** B ;

Algorithme 9 : CSBsimplexe$(A, \mathcal{L})$

Le théorème 3.8 justifie que les représentants positifs des vecteurs creux que nous recherchons sont des sommets du polyèdre des solutions réalisable de $\mathcal{S}_{LP}(A, g)$, pour certaines fonctions

5. Cette matrice se remplit au fur et à mesure par des vecteurs lignes augmentant son rang.

d'objectifs g. Or, nous ne savons pas quelles fonctions d'objectifs g permettent de trouver ces représentants positifs de vecteurs les plus creux, bien que la proposition 3.10 et le théorème 3.6 garantissent leur existence. Ce point constitue la difficulté majeure du choix des fonctions d'objectifs g à utiliser.

Ces fonctions g sont, d'après la proposition 3.10 et le théorème 3.6, sous la forme $g(x) = \sum_{i=1}^{n} \alpha_i(y_i - y_{n+i})$ avec $\alpha_i \in \{-1, 0, 1\}$. Il y 3^n telles fonctions d'objectifs. Cependant, les expérimentations réalisées à la section 3.4 montreront que se restreindre à des fonctions d'objectifs simples de la forme $f(x) = x_i$ et $f(x) = x_i \pm x_j$ suffisent dans la plupart des cas.

L'algorithme CSBsimplexe prend en entrée une matrice A et une liste $\mathcal{L}$ de fonctions d'objectifs de $\mathbb{Q}^{2n}$ dans $\mathbb{Q}$, définie par l'utilisateur. Nous appliquons l'algorithme du simplexe pour chacune de ces fonctions d'objectifs et nous récupérons chaque solution optimale x de $\mathcal{S}_{NLP}(A, f)$ sous la forme de vecteurs lignes dans une liste $\mathcal{V}$. Enfin, à partir de ces solutions optimales, nous construisons une base B qui sera une base la plus creuse de l'espace vectoriel généré par les vecteurs de $\mathcal{V}$ (mais cette base n'est pas nécessairement une base la plus creuse de $\mathrm{Ker}({}^tM)$).

Preuve – CSBsimplexe est correct. L'algorithme InitialisationSimplexe initialise le simplexe. L'ensemble des solutions $LP(A, g)$ étant borné, l'algorithme du simplexe renvoie un sommet x (solution optimale). Pour que x soit équivalent à une solution optimale de $\mathcal{S}_{NLP}(A, f)$, il faut vérifier $\zeta_{max} \neq 0$ d'après le théorème 3.6. Enfin, les dernières lignes de l'algorithme CSBsimplexe permettent de construire une base à partir des x de $\mathcal{V}$. L'algorithme s'arrête trivialement. □

Nous avons présenté l'algorithme CSBsimplexe, qui est l'algorithme principal de ce chapitre, ainsi que la difficulté du choix des fonctions d'objectifs à mettre dans $\mathcal{L}$ pour renvoyer une base B la plus creuse possible. Dans la section suivante, nous discutons de la complexité de l'algorithme CSBsimplexe et de son implantation.

3.3.4 Complexité et implantation

Complexité

Soient $\mathcal{L}$ une liste de fonctions d'objectifs et $\ell = \mathrm{card}(\mathcal{L})$ son nombre d'éléments. L'algorithme CSBsimplexe va résoudre ℓ programmes linéaires. La question de savoir si les programmes linéaires se résolvent facilement ou non mériterait d'être étudiée. En effet, dans certaines situations particulières, l'algorithme du simplexe a une complexité exponentielle [36].

En supposant que ces programmes linéaires se résolvent facilement, le temps de calcul sera plus ou moins proportionnel à ℓ. En pratique, il est donc primordial de limiter le nombre d'objectifs. En effet, d'après la proposition 3.10, il faudrait une liste de $3^n - 1$ objectifs pour avoir la garantie d'une base la plus creuse. Pour des raisons d'efficacité, on peut se limiter à des objectifs simples (voir section 3.4).

Implantation

Nous avons choisi d'implanter l'algorithme CSBsimplexe avec Maple, principalement pour pouvoir comparer nos temps d'exécution avec ceux de la section 2.5, en utilisant l'algorithme du simplexe pour différentes fonctions d'objectifs. L'algorithme du simplexe a été recodé en Maple en calcul exact.

Amélioration

Lors de l'initialisation du simplexe avec l'algorithme InitialisationSimplexe, le calcul de T ne dépend pas de w. Chaque appel à la boucle "pour tout" recrée la même matrice T. On peut donc calculer T une seule fois avant cette boucle ainsi que $\mathcal{J}_B$ et ne calculer que w en fonction de T et g dans la boucle "pour tout".

3.4 Expérimentations

Nos expérimentations ont été réalisées sur les modèles de la base de données BioModels [1]. Parmi les modèles réarrangés de BioModels, nous avons sélectionné tous les modèles impliquant un seul compartiment, avec des coefficients stœchiométriques rationnels et entiers.

Les transposées des matrices de stœchiométrie de chaque modèle sont les matrices A des programmes linéaires $\mathcal{S}_{LP}(A, g)$ que nous considérerons, pour différentes fonctions d'objectifs g.

Parmi ces modèles, nous considérons dans cette section les mêmes modèles que la section 2.5 (il y en a 74) et nous mettons de côté les 155 autres modèles.

Soient les listes de fonctions d'objectifs

$$\mathcal{L}_1 = \{f : \mathbb{Q}^n \to \mathbb{Q} \text{ telles que } f(x) = x_i \text{ pour } i \in [\![1, n]\!]\} \tag{3.13}$$

et

$$\mathcal{L}_2 = \mathcal{L}_1 \cup \{f : \mathbb{Q}^n \to \mathbb{Q} \text{ telles que } f(x) = x_i \pm x_j \text{ pour } i, j \in [\![1, n]\!] \text{ avec } i > j\}\,. \tag{3.14}$$

Dans les sections qui suivent, nous considérons les listes de fonctions d'objectifs $\mathcal{L}_1$ et $\mathcal{L}_2$ et nous donnons une synthèse des résultats obtenus en lançant, pour les matrices de stœchiométrie M de chacun des 74 modèles considérés dans cette section, CSBsimplexe$({}^tM, \mathcal{L}_1)$ et CSBsimplexe$({}^tM, \mathcal{L}_2)$. Soient $B_{\mathcal{L}_1}$ et $B_{\mathcal{L}_2}$ les bases obtenues avec CSBsimplexe pour les listes de fonctions d'objectifs $\mathcal{L}_1$ et $\mathcal{L}_2$ pour chaque matrice M. On note t_1 et t_2 les temps d'exécution en secondes de l'algorithme pour chacune de ces listes de fonctions d'objectifs.

3.4.1 Liste de fonctions d'objectifs $\mathcal{L}_1$

Il y a exactement n fonctions d'objectifs dans la liste $\mathcal{L}_1$. Ces fonctions d'objectifs $f(x) = x_k$ sont simples, et ont motivé ce chapitre. En effet, nous avons expliqué dans la section 3.1 que maximiser un coefficient devrait pousser un maximum d'autres coefficients à être nul.

La figure 3.1 donne, pour chaque modèle, le temps de calcul t_1 de CSBsimplexe$({}^tM, \mathcal{L}_1)$, les nombres de nonzéros d'une base la plus creuse B' de $\mathrm{Ker}({}^tM)$ (calculée avec CSB) et de la base $B_{\mathcal{L}_1}$ calculée par CSBsimplexe$({}^tM, \mathcal{L}_1)$, ainsi que les nombres de lignes m et $m_{\mathcal{L}_1}$ respectivement des bases B' et $B_{\mathcal{L}_1}$.

Excepté pour les modèles BIOMD038, BIOMD066 et BIOMD489 où il manque une loi de conservation par le calcul de CSBsimplexe$({}^tM, \mathcal{L}_1)$, les bases calculées ont la bonne dimension (i.e. $m_{L_1} = m$) mais ne sont pas nécessairement les plus creuses.

Parmi les 71 modèles pour lesquels CSBsimplexe calcule une base complète, 58 bases sont les plus creuses et 13 bases ne le sont pas. Sur ces 13 bases, la plus grande différence $\mathcal{N}(B_{\mathcal{L}_1}) - \mathcal{N}(B')$ vaut 6.

En comparant les temps d'exécution de la figure 3.1 avec ceux de la figure 2.1, on remarque que l'algorithme du simplexe avec $\mathcal{L}_1$ sur les modèles de la base de données BioModels s'avère souvent très efficace pour calculer des bases les plus creuses.

Modèle	m	$m_{\mathcal{L}_1}$	n	$\mathcal{N}(B')$	$\mathcal{N}(B_{\mathcal{L}_1})$	t_1
BIOMD013	6	6	27	33	35	0.78
BIOMD019	15	15	61	90	90	21.83
BIOMD038	7	6	17	28	22	0.15
BIOMD066	6	5	11	16	20	0.5
BIOMD086	5	5	17	32	37	0.20
BIOMD152	11	11	64	92	94	31.43
BIOMD153	11	11	75	111	112	54.81
BIOMD175	24	24	118	188	188	214.13
BIOMD186	5	5	11	18	18	0.6
BIOMD220	18	18	58	79	79	8.79
BIOMD223	20	20	84	97	97	42.98
BIOMD332	16	16	78	140	140	59.8
BIOMD333	12	12	54	97	97	13.51
BIOMD334	13	13	73	108	108	39.69
BIOMD362	10	10	34	52	52	2.37
BIOMD452	23	23	109	134	134	167.13
BIOMD475	7	7	23	40	40	0.90
BIOMD478	11	11	33	47	47	2.68
BIOMD489	20	19	53	59	56	6.83
BIOMD494	16	16	80	70	70	56.614
BIOMD574	4	4	195	317	317	214.13
BIOMD578	8	8	76	88	91	45.49

FIGURE 3.1 – Résultats après application de CSBsimplexe sur $\mathcal{L}_1$ sur un échantillon de modèles avec les bases (par lignes) $B' = \mathsf{CSB}(B)$ de dimension $m \times n$ et $B_{\mathcal{L}_1} = \mathsf{CSBsimplexe}({}^tM, \mathcal{L}_1)$ de dimension $m_{\mathcal{L}_1} \times n$, $\mathcal{N}(B')$ et $\mathcal{N}(B_{\mathcal{L}_1})$ leurs nombres de nonzéros, et t_1 le temps d'exécution de $\mathsf{CSBsimplexe}({}^tM, \mathcal{L}_1)$.

Remarque 3.5. *Nous avons écarté de nos calculs les 155 modèles qui, dans le chapitre 2, sont déjà les plus creux après un calcul de noyau (via la fonction NullSpace de* MAPLE*). L'algorithme CSB détecte en effet rapidement qu'elles sont les plus creuses. On montre également que l'on calcule pour tous ces modèles une base la plus creuse avec* $\mathsf{CSBsimplexe}({}^tM, \mathcal{L}_1)$.

3.4.2 Liste de fonctions d'objectifs $\mathcal{L}_2$

Pour former $\mathcal{L}_2$, nous avons ajouté à $\mathcal{L}_1$ les fonctions d'objectifs $f(x) = x_i \pm x_j$. Il y a donc exactement n^2 fonctions d'objectifs dans la liste $\mathcal{L}_2$. Ces nouvelles fonctions d'objectifs ont été ajoutées dans le but de rendre les bases calculées complètes et plus creuses.

La figure 3.2 donne, pour chaque modèle, le temps de calcul t_2 de $\mathsf{CSBsimplexe}({}^tM, \mathcal{L}_2)$, les nombres de nonzéros d'une base la plus creuse B' de $\mathrm{Ker}({}^tM)$ (calculée avec CSB) et de la base $B_{\mathcal{L}_2}$ calculée par $\mathsf{CSBsimplexe}({}^tM, \mathcal{L}_2)$, ainsi que le nombre de lignes m de la base B'.

On remarque que la liste de fonctions d'objectifs $\mathcal{L}_2$ suffit à obtenir une base complète de $\mathrm{Ker}({}^tM)$ pour tous nos modèles, contrairement à $\mathcal{L}_1$. Les nombres de lignes m et $m_{\mathcal{L}_2}$ de B' et de $B_{\mathcal{L}_2}$ sont donc égaux. Par ailleurs, comme $m = m_{\mathcal{L}_2}$, nous n'affichons pas la colonne $m_{\mathcal{L}_2}$ dans la figure 3.2.

Remarque 3.6. *Bien que les fonctions d'objectifs de* $\mathcal{L}_2$ *suffisent à obtenir des bases complètes pour tous les modèles de BioModels que nous avons étudiés, il existe des matrices sur lesquelles* $\mathcal{L}_2$ *n'est pas suffisant (voir exemple 3.12).*

Modèle	m	n	$\mathcal{N}(B')$	$\mathcal{N}(B_{\mathcal{L}_2})$	t_2
BIOMD013	6	27	33	35	26.74
BIOMD019	15	61	90	90	1032.82
BIOMD038	7	17	28	28	2.58
BIOMD066	6	11	20	20	0.54
BIOMD086	5	17	32	32	3.16
BIOMD152	11	64	92	93	1529.86
BIOMD153	11	75	111	112	3921.60
BIOMD175	24	118	188	188	27651.43
BIOMD186	5	11	18	18	0.59
BIOMD220	18	58	79	79	582.20
BIOMD223	20	84	97	97	4096.60
BIOMD332	16	78	140	140	4342.49
BIOMD333	12	54	97	97	573.12
BIOMD334	13	73	108	108	2654.43
BIOMD362	10	34	52	52	89.9
BIOMD452	23	109	134	134	16579.33
BIOMD475	7	23	40	40	18.62
BIOMD478	11	33	47	47	67.23
BIOMD489	20	53	59	59	263.58
BIOMD574	4	195	317	317	-
BIOMD578	8	76	88	88	3886.14

FIGURE 3.2 – Résultats après application de CSBsimplexe sur $\mathcal{L}_2$ sur un échantillon de modèles avec les bases (par lignes) $B' = \mathsf{CSB}(B)$ de dimension $m \times n$ et $B_{\mathcal{L}_2} = \mathsf{CSBsimplexe}({}^tM, \mathcal{L}_2)$ de dimension $m \times n$, $\mathcal{N}(B')$ et $\mathcal{N}(B_{\mathcal{L}_2})$ leurs nombres de nonzéros, et t_2 le temps d'exécution de $\mathsf{CSBsimplexe}({}^tM, \mathcal{L}_2)$.

Exemple 3.12. *Soit* $A = \begin{pmatrix} 0 & 0 & 2 & -1 & 0 \\ 2 & 0 & 2 & 2 & 1 \\ 2 & -1 & 2 & 2 & -2 \end{pmatrix}$. *Une base la plus creuse de* $Ker(A)$ *est* $B = \begin{pmatrix} 1 & 2 & 0 & 0 & -2 \\ -3 & 12 & 1 & 2 & 0 \end{pmatrix}$. *On montre par le calcul que* $B_1/||B_1||_1$ *ne maximise aucun objectif issu de* $\mathcal{L}_2$. *On conclut donc que* $\mathcal{L}_2$ *ne suffit pas toujours pour calculer une base complète de* A.

Parmi les 74 bases calculées avec CSBsimplexe, 71 bases sont les plus creuses. Le calcul de bases creuses avec $\mathcal{L}_2$ sur les modèles BIOMD013, BIOMD152 et BIOMD153 ne donne pas de bases les plus creuses. La plus grande différence $\mathcal{N}(B_{\mathcal{L}_2}) - \mathcal{N}(B')$ vaut 2 pour le modèle BIOMD013, cette différence vaut 1 pour les deux autres modèles.

3.4.3 Pré-traitement par CSBsimplexe avec $\mathcal{L}_1$ pour CSB

Utiliser la liste d'objectifs $\mathcal{L}_1$ pour pré-calculer une base creuse, en la complétant si nécessaire, puis appliquer CSB sur la base creuse ainsi obtenue permet dans le cas de certains modèles (souvent les plus gros modèles) de diminuer le nombre d'appels à EnhanceBasis dans CSB et de diminuer le temps de calcul d'une base la plus creuse.

La figure 3.3 donne, pour certain modèles, les temps d'exécution t_1 de $\mathsf{CSBsimplexe}({}^tM, \mathcal{L}_1)$, $t(v_2')$ de $\mathsf{CSB}(B)$ (version v_2') et t obtenu par application de $B_{\mathcal{L}_1} = \mathsf{CSBsimplexe}({}^tM, \mathcal{L}_1)$ suivie de $\mathsf{CSBprime}(B_{\mathcal{L}_1})$ (version v_2').

Excepté pour le modèle BIOMD153, les bases calculées avec $\mathsf{CSBsimplexe}({}^tM, \mathcal{L}_1)$ sont les

Modèle	m	n	t_1	$t(v'_2)$	t
BIOMD153	11	75	54.81	75.58	71.19
BIOMD175	24	118	214.13	13357.17	3636.91
BIOMD332	16	78	59.8	15226.30	350.99
BIOMD333	12	54	13.51	371.66	35.97
BIOMD475	7	23	0.9	14.71	4.89

FIGURE 3.3 – Temps d'exécution de calculs d'une base la plus creuse avec CSBsimplexe puis CSB
avec $m \times n$ la dimension des bases calculées, t_1 le temps d'exécution de CSBsimplexe$({}^tM, \mathcal{L}_1)$, $t(v'_2)$ le temps d'exécution de la version v'_2 de CSB (voir section 2.5) et t le temps d'exécution du calcul d'une base la plus creuse par l'algorithme par application de $B_{\mathcal{L}_1}$ = CSBsimplexe$({}^tM, \mathcal{L}_1)$ suivie de CSBprime$(B_{\mathcal{L}_1})$ (version v'_2)

plus creuses possibles. Pour ces modèles, l'appel à CSB sur $B_{\mathcal{L}_1}$ permet de garantir qu'il n'y a pas de base la plus creuse de $B_{\mathcal{L}_1}$. Pour le modèle BIOMD153, où $\mathcal{N}(B_{\mathcal{L}_1}) - \mathcal{N}(B') = 1$, l'appel à CSB sur $B_{\mathcal{L}_1}$ effectue deux appels à EnhanceBasis. Le premier appel à EnhanceBasis calcule une base la plus creuse en diminuant le nombre de nonzéro de la base de 1. Le second appel à EnhanceBasis garantit que la base calculées au premier appel est une base la plus creuse.

Nous avons testé ce procédé sur les modèles BIOMD205 et BIOMD457 qui avaient été écartés de l'analyse de données dans la section 2.5, car les calculs avaient été abandonnés après une semaine de calcul. Même en appliquant un pré-calcul par CSBsimplexe, CSB n'est pas capable de calculer des bases les plus creuses en moins d'une semaine pour ces deux modèles.

Le pré-traitement par calcul d'une base creuse par CSBsimplexe$({}^tM, \mathcal{L}_1)$ pour les deux plus grands modèles que nous considérons dans BioModels (les modèles BIOMD175 et BIOM332), montre son efficacité. Le temps de calcul d'une base la plus creuse du modèle BIOMD175 a été divisé par 3.67. Le temps de calcul d'une base la plus creuse du modèle BIOMD332 a été divisé par 43.38.

3.4.4 Calculs sur des modèles trop volumineux pour CSB

Comme dans la section 2.5, les calculs sur les bases des modèles BIOMD205 (de dimension 205×194) et BIOMD457 (de dimension 141×166) ne terminent pas après un semaine d'exécution, en appliquant $B_{\mathcal{L}_1}$ = CSBsimplexe$({}^tM, \mathcal{L}_1)$ puis CSBprime$(B_{\mathcal{L}_1})$ (version v'_2).

Bien que CSB échoue à calculer une base la plus creuse, nous pouvons donner les bases creuses calculées avec CSBsimplexe pour ces deux modèles, sans garantie qu'elles soient effectivement les plus creuses possible. La figure 3.4 donne les temps d'exécution t_1 de CSBsimplexe pour ces deux modèles ainsi que les nombres de nonzéros des bases calculées avec CSBsimplexe et le pré-traitement de CSB.

Modèle	m	n	$\mathcal{N}(B)$	$\mathcal{N}(B^P)$	$\mathcal{N}(B_{\mathcal{L}_1})$	t_1
BIOMD205	36	194	668	497	362	2099.45
BIOMD457	25	166	466	330	314	1486.68

FIGURE 3.4 – Gros modèles
avec $m \times n$ la dimension des bases calculées, B base de ${}^tM$ calculée avec la fonction NullSpace de MAPLE, B^P base calculée par les pré-traitements de CSB sur B, $B_{\mathcal{L}_1}$ = CSBsimplexe$({}^tM, \mathcal{L}_1)$, t_1 le temps d'exécution du calcul de $B_{\mathcal{L}_1}$ et $\mathcal{N}(B)$, $\mathcal{N}(B^P)$, $\mathcal{N}(B_{\mathcal{L}_1})$ nombres de nonzéros

Les bases creuses $B_{\mathcal{L}_1}$ calculées pour les modèles BIOMD205 et BIOMD457 sont plus creuses que les pré-traitements de CSB. Bien que CSBsimplexe ne soit pas garanti, il semble pertinent de l'utiliser pour de grands modèles.

Chapitre 4

Simplification de systèmes paramétriques d'équations

Sommaire

4.1 Introduction 58
4.2 Fractions rationnelles 59
4.2.1 Représentation matricielle d'une fraction 59
4.2.2 Application monomiale 60
4.2.3 Action d'une application monomiale 60
4.2.4 Base d'un espace vectoriel modulo un espace vectoriel 61
4.2.5 L'algorithme CSBmodulo 61
4.2.6 Preuve de CSBmodulo 63
4.2.7 L'algorithme getSparsestFraction 65
4.3 Sommes de fractions rationnelles 69
4.3.1 Représentation matricielle d'une somme de fractions 69
4.3.2 Action d'une application monomiale 70
4.3.3 Algorithme getSparsestSumOfFractions 71
4.3.4 Application à la simplification de systèmes d'équations 72
4.3.5 Complexité et implantation 73

Ce chapitre a fait l'objet d'une publication à CASC2016. Il présente de nouveaux algorithmes pour calculer des représentations creuses de systèmes paramétriques de fractions rationnelles via des changements de variables.

Le but de ces algorithmes est d'aider l'analyse de systèmes paramétriques de fractions rationnelles en produisant des formulations plus creuses et équivalentes. Simplifier des systèmes paramétriques est une tâche centrale, vu que de nombreuses méthodes (telles que l'analyse de points fixes, l'analyse de bifurcation, ...) reposent sur des calculs plutôt coûteux en géométrie algébrique réelle (voir [4] et les références qui y sont incluses).

Des approches basées sur les symétries [40, 53, 33, 25, 47] réduisent le nombre de paramètres et en conséquence simplifient généralement l'analyse des systèmes paramétriques. Contrairement à ces approches, la nôtre conserve le nombre de paramètres (dans le même esprit que [40, Algorithm SemiRectifySteadyPoints]) et rend le système le plus creux possible (au sens des algorithmes getSparsestFraction et getSparsestSumOfFractions donnés aux sections 4.2.7 et 4.3.3).

4.1 Introduction

Ce travail a été motivé par le système différentiel suivant :

$$\begin{cases} G'(t) = \theta(1 - G(t)) - \alpha k_1 k_2 k_3 P(t)^4 G(t) \\ M'(t) = \rho_b(1 - G(t)) + \rho_f G(t) - \delta_M M(t) \\ P'(t) = \dfrac{4\theta(1 - G(t)) - 4\alpha k_1 k_2 k_3 P(t)^4 G(t) - \delta_P P(t) + \beta M(t)}{16 k_1 k_2 k_3 P(t)^3 + 9 k_1 k_2 P(t)^2 + 4 k_1 P(t) + 1} \end{cases} \tag{4.1}$$

dans lequel les fonctions inconnues sont G, M et P, et où les paramètres sont θ, α, k_1, k_2, k_3, ρ_b, ρ_f, δ_M, δ_P et β.

L'équation (4.1) apparaissait dans [16], et a été réécrite avec le changement de variables deviné $\bar{k}_3 = k_1 k_2 k_3$, $\bar{k}_2 = k_1 k_2$, $\bar{k}_1 = k_1$, afin d'obtenir [16, Equation (3.4)] (dans le cas où $n = 4$ et $\gamma_0 = 1$).

Notre nouvel algorithme getSparsestSumOfFractions (voir section 4.3.3) a été conçu pour calculer automatiquement de tels changements de variables, produisant le système suivant, plus simple :

$$\begin{cases} G'(t) = \theta(1 - G(t)) - \alpha \bar{k}_3 P(t)^4 G(t) \\ M'(t) = \rho_b(1 - G(t)) + \rho_f G(t) - \delta_M M(t) \\ P'(t) = \dfrac{4\theta(1 - G(t)) - 4\alpha \bar{k}_3 P(t)^4 G(t) - \delta_P P(t) + \beta M(t)}{16 \bar{k}_3 P(t)^3 + 9 \bar{k}_2 P(t)^2 + 4 \bar{k}_1 P(t) + 1}. \end{cases} \tag{4.2}$$

La légère diminution du degré entre (4.1) et (4.2) (grâce au changement de variables deviné) montre son utilité lors de la recherche d'une bifurcation de Hopf. En effet, appliquer le critère de Routh-Hurwitz sur (4.1) et (4.2) débouche sur des systèmes semi-algébriques de la forme $h_1, h_2, h_3, h_4 = 0$, $h_5, h_6, h_7 > 0$, avec les degrés respectifs 9, 2, 9, 42, 12, 20, 23 pour le système (4.1), et les degrés plus petits 7, 2, 7, 32, 8, 14, 19 pour le système (4.2).

Les techniques présentées dans ce chapitre reposent sur une observation simple. On considère un système paramétrique de fractions rationnelles, dans lequel les paramètres sont des éléments d'un ensemble U. On associe à ce système ce que nous nommons une *représentation matricielle* qui code dans une matrice les degrés des monômes en U pour chaque paramètre. Nous considérons ensuite une application monomiale ϕ (i.e. une application qui envoie chaque paramètre de U vers un monôme en les éléments d'un ensemble $\bar{U}$). On montre que l'application monomiale ϕ agit linéairement sur la représentation matricielle, nous permettant de rechercher une représentation matricielle la plus creuse possible en utilisant uniquement des techniques d'algèbre linéaire.

Cependant, l'usage de fractions amène quelques difficultés puisque les fractions sont invariantes lorsque l'on multiplie simultanément le numérateur et le dénominateur par la même valeur. L'exemple 4.23 montre comment réécrire automatiquement la fraction $\dfrac{x}{1 + x + ix\tau_1 w}$ en la fraction plus creuse $\dfrac{1}{y + 1 + i\tau_1 w}$, en divisant d'abord à la fois le numérateur et le dénominateur par x et ensuite en posant $y = 1/x$.

La section 4.2 introduit les concepts basiques et deux nouveaux algorithmes permettant de simplifier une fraction rationnelle. Le premier algorithme, nommé CSBmodulo calcule une représentation la plus creuse possible d'un espace vectoriel modulo un autre espace vectoriel, explicités par leurs bases respectives. Le deuxième algorithme, nommé getSparsestFraction est une application directe de l'algorithme CSBmodulo pour calculer une représentation la plus creuse d'une fraction. La section 4.3 décrit comment généraliser les concepts de la section 4.2

pour des sommes de fractions rationnelles ainsi que l'algorithme getSparsestSumOfFractions qui généralise l'algorithme getSparsestFraction. Une application de cet algorithme pour des systèmes paramétriques d'équations sera également présenté dans la section 4.3.4.

4.2 Fractions rationnelles

Soient $\mathbb{K}$ un corps commutatif et $U = \{u_1, u_2, \ldots, u_n\}$ un ensemble de variables. Dans ce chapitre, nous considérons des monômes en U de la forme $u_1^{\alpha_1} \cdots u_n^{\alpha_n}$ où les α_i sont dans $\mathbb{Z}$. Comme les exposants entiers négatifs sont permis,, $u_1 u_3/u_2$ est considéré comme un monôme. Le vecteur ligne $\begin{pmatrix} 1 & \cdots & 1 \end{pmatrix}$ de longueur ℓ est noté $\mathbb{1}_\ell$ (ou simplement $\mathbb{1}$ lorsque le contexte est clair).

4.2.1 Représentation matricielle d'une fraction

Définition 4.21. *Soient une matrice $N = (\alpha_{i,j})$ dans $\mathbb{Z}^{n\times\ell}$, un ensemble $T = \{t_1, \ldots, t_\ell\}$ d'éléments de $\mathbb{K}$, et un entier g tel que $1 \leq g < \ell$. Par définition, le triplet (N, T, g) écrit sous la forme*

$$\left(\begin{array}{ccc|ccc} \alpha_{1,1} & \cdots & \alpha_{1,g} & \alpha_{1,g+1} & \cdots & \alpha_{1,\ell} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \alpha_{n,1} & \cdots & \alpha_{n,g} & \alpha_{n,g+1} & \cdots & \alpha_{n,\ell} \\ t_1 & \cdots & t_g & t_{g+1} & \cdots & t_\ell \end{array}\right) \begin{matrix} u_1 \\ \vdots \\ u_n \end{matrix},$$

représente la fraction

$$q = \frac{\sum_{i=1}^{g} t_i m_i}{\sum_{i=g+1}^{\ell} t_i m_i} \tag{4.3}$$

où $m_i = u_1^{\alpha_{1,i}} u_2^{\alpha_{2,i}} \cdots u_n^{\alpha_{n,i}}$. Le triplet (N, T, g) (ou simplement N lorsque le contexte est clair) est appelé représentation matricielle de q.

Exemple 4.13. *Le triplet*

$$\left(\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \\ t_1 & t_2 & t_3 & t_4 \end{array}\right) \begin{matrix} a \\ b \end{matrix} \tag{4.4}$$

avec $g = 2$, $U = \{a, b\}$, $\mathbb{K} = \mathbb{Q}(x, y)$ et $t_1 = 2, t_2 = 3xy, t_3 = y^2, t_4 = 5x$ représente la fraction

$$q_1 = \frac{2a + 3b^2xy}{ay^2 + 5bx} \in \mathbb{K}(U). \tag{4.5}$$

Proposition 4.13. *Soit un triplet (N, T, g) représentant une fraction q. Alors, pour tout vecteur colonne $v \in \mathbb{Z}^n$, le triplet $(N + v\mathbb{1}, T, g)$ représente également la fraction q.*

Démonstration. Pour tout entier $\delta \in \mathbb{Z}$, ajouter $\delta\mathbb{1}$ à la k-ième ligne de N équivaut à multiplier à la fois le numérateur et le dénominateur de q par le même monôme u_k^δ. □

Exemple 4.14. *D'après la proposition 4.13 avec $v = (-1, 2)$, les triplets*

$$\left(\begin{array}{cc|cc} 1 & 2 & 1 & 1 \\ -2 & -2 & -2 & -1 \\ t_1 & t_2 & t_3 & t_4 \end{array}\right) \begin{matrix} \bar{a} \\ \bar{b} \end{matrix} \tag{4.6}$$

et

$$\left(\begin{array}{cc|cc} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right) \begin{matrix} \bar{a} \\ \bar{b} \end{matrix} \\ \begin{matrix} t_1 & t_2 & t_3 & t_4 \end{matrix} \tag{4.7}$$

représentent la même fraction $\bar{q}_1$ *de* $\mathbb{K}(\bar{a}, \bar{b})$ *écrite de deux façons différentes :*

$$\bar{q}_1 = \frac{2\frac{\bar{a}}{\bar{b}^2} + 3\frac{\bar{a}^2}{\bar{b}^2}xy}{\frac{\bar{a}}{\bar{b}^2}y^2 + 5\frac{\bar{a}}{\bar{b}}x} \tag{4.8}$$

et

$$\bar{q}_1 = \frac{2 + 3\bar{a}xy}{y^2 + 5\bar{b}x}. \tag{4.9}$$

4.2.2 Application monomiale

Définition 4.22. *Soient une matrice inversible* $C \in \mathbb{Q}^{n\times n}$ *et deux ensembles de variables* $U = \{u_1, u_2, \ldots, u_n\}$ *et* $\bar{U} = \{\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_n\}$*. La matrice* C *définit le morphisme d'anneaux* ϕ^C *de* $\mathbb{K}(U)$ *vers* $\mathbb{K}(\bar{U}^{1/p})$*, où* p *est le* PPCM *de tous les dénominateurs des éléments de* C*, dans le sens suivant :*

$$\phi^C(u_k) \;=\; \prod_{i=1}^{n} \bar{u}_i^{c_{i,k}} \text{ pour tout } k \in [\![1, n]\!].$$

Le morphisme ϕ^C *est appelé une* application monomiale*. On note simplement* ϕ *au lieu de* ϕ^C *lorsqu'il n'y a pas de confusion possible.*

Dans cette partie, nous considérons des applications monomiales ϕ et des fractions q particulières telles que $\phi(q)$ est une fraction rationnelle de $\mathbb{K}(\bar{U})$.

Exemple 4.15. *La matrice*

$$\left(\begin{array}{cc} \alpha & \gamma \\ \beta & \delta \end{array}\right) \begin{matrix} \bar{a} \\ \bar{b} \end{matrix} \\ \begin{matrix} a & b \end{matrix}$$

définit l'application monomiale ϕ *définie par* $\phi(a) = \bar{a}^\alpha \bar{b}^\beta$ *et* $\phi(b) = \bar{a}^\gamma \bar{b}^\delta$*.*

4.2.3 Action d'une application monomiale

Proposition 4.14. *Soient un triplet* (N, T, g) *représentant une fraction* q *de* $\mathbb{K}(U)$ *et une matrice inversible* C *de* $\mathbb{Q}^{n\times n}$*. Si* CN *contient uniquement des éléments de* $\mathbb{Z}$*, alors le triplet* (CN, T, g) *représente la fraction* $\phi^C(q)$ *de* $\mathbb{K}(\bar{U})$*.*

Démonstration. Immédiat. □

Corollaire 4.5. *Soient un triplet* (N, T, g) *représentant une fraction* q*, une matrice inversible* C *de* $\mathbb{Q}^{n\times n}$ *et un vecteur colonne* v *de* $\mathbb{Q}^n$*. Si* $\bar{N} = CN + v\mathbb{1}$ *contient uniquement des éléments de* $\mathbb{Z}$*, alors* $\bar{N}$ *est une représentation matricielle de* $\phi^C(q)$*.*

Démonstration. Il s'agit d'une conséquence directe des propositions 4.13 et 4.14. □

Exemple 4.16. *Soit la matrice inversible* C

$$\begin{pmatrix} 1 & 1 \\ -2 & -1 \\ a & b \end{pmatrix} \begin{matrix} \bar{a} \\ \bar{b} \end{matrix} . \tag{4.10}$$

Soient N *et* $\bar{N}$ *les matrices respectivement issues des équations* (4.4) *et* (4.7) *représentant les fractions* q_1 *and* $\bar{q}_1$ *des exemples 4.13 et 4.14. On vérifie aisément que* $\bar{N} = CN + v\mathbb{1}$*, où* $v = (-1, 2)$*. Par conséquent, le corollaire 4.5 implique que* $\bar{q}_1$ *est bien égal à* $\phi^C(q_1)$*.*

4.2.4 Base d'un espace vectoriel modulo un espace vectoriel

Soit un triplet (N, T, g) représentant une fraction $q \in \mathbb{K}(U)$ avec les notations de la définition 4.21. Dire qu'une matrice N est creuse signifie que de nombreux monômes m_i contiennent peu d'éléments u_j. Cela implique que la fraction q est creuse par rapport aux u_j.

Dans ce chapitre, nous avons choisi de rendre la matrice N la plus creuse possible afin de simplifier la fraction q correspondante. Pour ce faire, nous nous autorisons à réaliser des changements de variables monomiaux sur U. Plus précisément, en nous appuyant sur le corollaire 4.5, nous recherchons une matrice inversible C de $\mathbb{Q}^{n\times n}$ et un vecteur ligne v de $\mathbb{Q}^n$, tels que la matrice $\bar{N} = CN + v\mathbb{1}$ possède uniquement des coefficients entiers et est la plus creuse. La matrice C et le vecteur v donnés dans l'exemple 4.16 anticipent les algorithmes qui seront donnés aux sections 4.2.5 et 4.2.7. Ces algorithmes permettront de calculer une matrice la plus creuse possible $\bar{N}$.

Définition 4.23. *Soient* $N \in \mathbb{Q}^{n\times\ell}$ *et* $P \in \mathbb{Q}^{s\times\ell}$ *deux matrices de rang plein selon les lignes. On suppose que* $\begin{pmatrix} N \\ P \end{pmatrix}$ *est de rang plein selon les lignes. On appelle* base la plus creuse de N modulo P *toute matrice* N' *qui est la plus creuse et satisfait* $N' = CN + VP$ *pour une matrice inversible* C *et une matrice* V*.*

Proposition 4.15. *Soit une base la plus creuse* N' *de* N *modulo* P *avec les mêmes hypothèses qu'à la définition 4.23. Alors, il existe des matrices* P', C', V', G', W' *telles que* $\begin{pmatrix} N' \\ P' \end{pmatrix}$ *soit une base la plus creuse de la matrice* $\begin{pmatrix} N \\ P \end{pmatrix}$*, avec* $\begin{pmatrix} N' \\ P' \end{pmatrix} = \begin{pmatrix} C' & V' \\ G' & W' \end{pmatrix} \begin{pmatrix} N \\ P \end{pmatrix}$ *où* C' *est inversible.*

Démonstration. L'existence de C' et de V' est donnée par la définition 4.23. Soit la matrice $M' = \begin{pmatrix} N' \\ P \end{pmatrix}$. Si M' n'est pas une base la plus creuse de $\begin{pmatrix} N \\ P \end{pmatrix}$, elle peut être rendue plus creuse en utilisant le corollaire 2.1. De plus, la ligne à améliorer ne peut pas être une ligne de N', puisque N' est une base la plus creuse de N modulo P. Après avoir appliqué le corollaire 2.1 un certain nombre de fois, on obtient une base la plus creuse $\begin{pmatrix} N' \\ P' \end{pmatrix}$ de $\begin{pmatrix} N \\ P \end{pmatrix}$ qui prouve l'existence de P', G' et W'. □

4.2.5 L'algorithme CSBmodulo

L'algorithme CSB (Compute Sparsest Basis), décrit à la section 2.3, résout le problème de la section 4.2.4 avec le cas particulier où v est le vecteur nul.

En effet, l'algorithme CSB prend en entrée une matrice M de rang plein selon les lignes et renvoie une matrice la plus creuse possible (i.e. avec le moins de nonzéros) $\bar{M}$ à coefficients dans

$\mathbb{Z}$, qui est ligne-équivalente à M (on rappelle que deux matrices A et B de même dimensions sont dites lignes-équivalentes si $A = PB$ pour une matrice inversible P).

Par conséquent, on doit généraliser l'algorithme CSB pour calculer une base la plus creuse des lignes de N "modulo" une autre base ; c'est ce que fait l'algorithme CSBmodulo.

L'algorithme CSBmodulo ci-dessous prend en entrée deux matrices N et P telles que $\begin{pmatrix} N \\ P \end{pmatrix}$ soit de rang plein selon les lignes. Il renvoie une matrice $\bar{N}$ et une matrice inversible $C \in \mathbb{Q}^{n \times n}$ telles que $\bar{N} = CN + VP$ avec une matrice V de $\mathbb{Q}^{n \times s}$. Par ailleurs, $\bar{N}$ est la plus creuse possible et ses coefficients sont dans $\mathbb{Z}$. La preuve de l'algorithme CSBmodulo est donnée à la section 4.2.6.

L'idée de l'algorithme CSBmodulo est la suivante. On considère d'abord N et P de façon symétrique en construisant une base la plus creuse de $M = \begin{pmatrix} N \\ P \end{pmatrix}$ à la ligne 2, et on obtient ainsi une matrice $\bar{M}$, dont les lignes sont triées par nombre croissant de nonzéros à la ligne 4. Par conséquent, $\bar{M} = DM$ pour une matrice inversible D. Ensuite, la matrice $\bar{N}$ est obtenue en choisissant n lignes $\bar{M}_{r_1}, \ldots, \bar{M}_{r_n}$ de $\bar{M}$. En notant E la matrice composée des n premières colonnes de D, pour n'importe quel choix de lignes dans $\bar{M}$, on obtient $\bar{N} = CN + VP$ où C est composée des lignes $E_{r_1}, \ldots, E_{r_n}$. Le choix des lignes doit être effectué avec soin. Premièrement, la matrice C doit être inversible. Cette condition est garantie par la ligne 9. Deuxièmement, la matrice $\bar{N}$ doit être la plus creuse possible. Cette condition est garantie par la sélection des premières lignes les plus creuses possibles (i.e. les premières lignes de $\bar{M}$ puisque ses lignes sont triées).

Exemple 4.17. *Considérons les matrices suivantes de rang plein selon les lignes :*

$$N = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

et

$$P = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

qui seront utilisées plus loin dans l'exemple 4.25.

Les matrices D et $\bar{M}$ calculées par les lignes 2–5 de l'algorithme CSBmodulo sont (en utilisant notre implantation de CSB)

$$\bar{M} = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

et

$$D = \begin{pmatrix} 1 & 1 & -1 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 1 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Entrée : Deux matrices $N \in \mathbb{Z}^{n\times\ell}$ et $P \in \mathbb{Z}^{s\times\ell}$ telles que $\begin{pmatrix} N \\ P \end{pmatrix}$ soit une matrice de rang plein selon les lignes.
Sortie : Une matrice $\bar{N} \in \mathbb{Z}^{n\times\ell}$ et une matrice inversible $C \in \mathbb{Q}^{n\times n}$ telles que $\bar{N} = CN + VP$ où V est une matrice de $\mathbb{Q}^{n\times s}$. De plus, $\bar{N}$ est la plus creuse possible.

1 **Début**
2 $M \leftarrow \begin{pmatrix} N \\ P \end{pmatrix}$;
3 $\bar{M} \leftarrow \mathsf{CSB}(M)$;
4 Trier les lignes de $\bar{M}$ par nombre croissant de nonzéros (du haut vers le bas) ;
5 Calculer la matrice inversible $D \in \mathbb{Q}^{(n+s)\times(n+s)}$ telle que $\bar{M} = DM$;
6 Noter $E \in \mathbb{Q}^{(n+s)\times n}$ la matrice formée par les n premières colonnes de D ;
7 Soient $C \in \mathbb{Q}^{0\times n}$ et $\bar{N} \in \mathbb{Z}^{0\times\ell}$ des matrices à zéro ligne ;
8 **Pour** i **allant de** 1 **à** $n+s$ **faire**
9 **Si** $\text{Rang}\begin{pmatrix} C \\ E_i \end{pmatrix} > \text{Rang}(C)$ **alors**
10 $C \leftarrow \begin{pmatrix} C \\ E_i \end{pmatrix}$;
11 $\bar{N} \leftarrow \begin{pmatrix} \bar{N} \\ \bar{M}_i \end{pmatrix}$;
12 **Renvoyer** $\bar{N}$, C ;

Algorithme 10 : $\mathsf{CSBmodulo}(N, P)$

L'extraction de la matrice $\bar{N}$ à partir de $\bar{M}$ opérée par les lignes 6–11 considère les trois premières colonnes de D et ignore les lignes 2, 4 et 6 de $\bar{M}$, ce qui nous ramène à

$$\bar{N} = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

Remarque 4.7. *L'algorithme CSBmodulo extrait la matrice C à partir des n premières colonnes de D. Cette extraction pourrait être réalisée en calculant le profil de rang par lignes (en anglais "row rank profile") [23] : le* profil de rang par lignes *d'une matrice A de dimension $m \times n$ et de rang r est la plus petite séquence lexicographique de r indices de lignes linéaires indépendantes de A. Un algorithme de décomposition $PLUQ$ utilisant l'élimination de Gauss est proposé dans [23] pour calculer un tel ensemble.*

4.2.6 Preuve de CSBmodulo

La preuve de l'algorithme CSBmodulo est assez technique, notamment la partie prouvant que la matrice $\bar{N}$ calculée est la plus creuse possible. Comme elle n'est pas nécessaire à la compréhension de l'algorithme ni de ce qui suit dans ce chapitre, le lecteur peut l'admettre et passer cette section.

Lemme 4.7. *Soient $H \in \mathbb{Q}^{m\times\ell}$ et $U' = \mathbb{Q}^{t\times m}$. Si $U'H = 0$, alors* $\text{Rang}(U') + \text{Rang}(H) \leq m$.

Démonstration. Conséquence directe du théorème du Rang appliqué sur la transposée de H. □

Preuve – CSBmodulo est correct. Le fait que les matrices calculées $\bar{N}, C$ satisfassent $\bar{N} = CN + VP$ pour une certaine matrice V est une conséquence de la stratégie de sélection dans la boucle. La justification de ce point est laissée au lecteur. De plus, la matrice $\bar{N}$ a ses coefficients dans $\mathbb{Z}$ puisqu'elle est une sous-matrice de $\bar{M}$, qui a également ses coefficients dans $\mathbb{Z}$ étant donné qu'elle est calculée par l'algorithme CSB.

Le point difficile est de montrer que $\bar{N}$ est la plus creuse possible. Pour le prouver, nous supposons que $\mathcal{N}(\bar{N}) > \mathcal{N}(N')$ pour une base la plus creuse N' de N modulo P, et nous allons en déduire une contradiction.

Par la proposition 4.15, il existe une matrice P' telle que $\begin{pmatrix} N' \\ P' \end{pmatrix}$ est une base la plus creuse de la matrice $\begin{pmatrix} N \\ P \end{pmatrix}$. On note M' la matrice obtenue en triant les lignes de $\begin{pmatrix} N' \\ P' \end{pmatrix}$ par nombre croissant de nonzéros. On introduit les indices $s_1 < \cdots < s_n$ tels que $M'_{s_i} = N'_i$.

D'après le corollaire 2.2, $\mathcal{N}(\bar{M}_i) = \mathcal{N}(M'_i)$ pour tout $i \in [\![1, n]\!]$, et il existe une matrice triangulaire inférieure par blocs et inversible T telle que $\bar{M} = TM'$.

Dans un souci de simplicité, on suppose que les nombres de nonzéros dans les lignes de $\bar{M}$ sont strictement croissants (quitte à trier les lignes), donc la matrice T est triangulaire inférieure avec des éléments diagonaux non nuls.

On note $r_1, \ldots, r_n$ les indices des lignes de $\bar{M}$ qui sont sélectionnées par la boucle dans l'algorithme CSBmodulo pour générer la matrice $\bar{N}$ (i.e. $\bar{N}_i = \bar{M}_{r_i}$ pour $i \in [\![1, n]\!]$).

Comme nous avons supposé que $\mathcal{N}(\bar{N}) > \mathcal{N}(N')$, alors $\mathcal{N}(\bar{N}_k) > \mathcal{N}(N'_k)$ pour un certain indice k. En prenant k minimal, on a $\mathcal{N}(\bar{N}_1) \leq \mathcal{N}(N'_1)$, …, $\mathcal{N}(\bar{N}_{k-1}) \leq \mathcal{N}(N'_{k-1})$ et $\mathcal{N}(\bar{N}_k) > \mathcal{N}(N'_k)$.

À partir des inégalités ci-dessus, on a $r_1 \leq s_1$, $r_2 \leq s_2$, …, $r_{k-1} \leq s_{k-1}$ et $r_k > s_k$, que l'on récapitule ici :

$$\begin{array}{ccc} \bar{M} & & M' \\ \begin{pmatrix} \vdots \\ \bar{M}_{r_1} \\ \vdots \\ \vdots \\ \bar{M}_{r_2} \\ \vdots \\ \bar{M}_{r_{k-1}} \\ \vdots \\ \vdots \\ \vdots \\ \bar{M}_{r_k} \\ \vdots \end{pmatrix} & = & \begin{pmatrix} T_{1,1} & & \\ \vdots & \ddots & \\ T_{n,1} & \cdots & T_{n,n} \end{pmatrix} \end{array} \begin{pmatrix} \vdots \\ \vdots \\ M'_{s_1} \\ \vdots \\ M'_{s_2} \\ \vdots \\ \vdots \\ M'_{s_{k-1}} \\ \vdots \\ M'_{s_k} \\ \vdots \\ \vdots \end{pmatrix}.$$

Parmi les s_k premières lignes de $\bar{M}$, il y a $k-1$ lignes qui ont été sélectionnées par l'algorithme. Par conséquent, $s_k - k + 1$ lignes n'ont pas été sélectionnées, impliquant que chaque ligne non sélectionnée E_i avec $i \leq s_k$ est une combinaison linéaire des lignes E_j précédentes avec $j < i$.

En stockant ligne par ligne les combinaisons linéaires ci-dessus dans une matrice U, on obtient une matrice de dimension $(s_k - k + 1) \times n$, dont les colonnes indicées de $s_k + 1$ à n sont nulles. De plus, la matrice U est de rang plein selon les lignes puisque U est sous forme échelonnée.

Écrivons $D = \begin{pmatrix} E & F \end{pmatrix}$. Par définition de U, on a $UD = \begin{pmatrix} UE & UF \end{pmatrix} = \begin{pmatrix} 0 & UF \end{pmatrix}$. Puisque $\bar{M} = DM$, alors $U\bar{M} = UDM = \begin{pmatrix} 0 & UF \end{pmatrix} \begin{pmatrix} N \\ P \end{pmatrix} = UFP$.

Puisque $\bar{M} = TM'$, alors $U\bar{M} = UTM'$. Puisque les colonnes de U indicées de $s_k + 1$ à n sont nulles, la matrice UT a également ses colonnes indicées de $s_k + 1$ à n nulles. De plus, UT est également de rang plein selon les lignes.

Avec les notations de la proposition 4.15 et quelques calculs simples, le produit UTM' peut être écrit sous la forme $UTHN + JP$ où H contient s_k lignes dont les lignes $1, 2, \ldots, k$ de C' et quelques lignes de G', pour une certaine matrice J.

Par conséquent, $U\bar{M} = UFP = UTHN + JP$. Cela implique que $(UF - J)P = (UTH)N$. Puisque la matrice $\begin{pmatrix} N \\ P \end{pmatrix}$ est de rang plein selon les lignes, les matrices UTH et $(UF - J)$ sont nécessairement nulles.

D'après le lemme 4.7, $\text{Rang}(UT) + \text{Rang}(H) \leq s_k$. Cependant, $\text{Rang}(UT) = s_k - k + 1$, et $\text{Rang}(H) \geq k$ puisque les k lignes $C'_1, \ldots, C'_k$ sont prises à partir de la matrice inversible C'. On a donc $\text{Rang}(UT) + \text{Rang}(H) \geq s_k + 1$, ce qui est en contradiction avec $\text{Rang}(UT) + \text{Rang}(H) \leq s_k$. Par conséquent, l'hypothèse $\mathcal{N}(\bar{N}) > \mathcal{N}(N')$ aboutit à une contradiction, d'où $\mathcal{N}(\bar{N}) \leq \mathcal{N}(N')$, d'où $\bar{N}$ est bien la plus creuse possible. □

4.2.7 L'algorithme getSparsestFraction

Cette section présente l'algorithme getSparsestFraction qui calcule une représentation la plus creuse possible d'une fraction q en utilisant une application monomiale. L'algorithme getSparsestFraction repose sur l'algorithme CSBmodulo.

Nous présentons d'abord les propositions 4.16 et 4.17 qui sont nécessaires lorsque le nombre de variables dans U peut être réduit par une application monomiale. Les propositions 4.16 et 4.17 sont en fait un cas particulier d'un traitement plus général basé sur une symétrie de type dilatation.

Proposition 4.16. *Soit un triplet (N, T, g) représentant une fraction $q \in \mathbb{K}(U)$. Si N n'est pas de rang plein selon les lignes, alors il existe une application monomiale ϕ^C telle que (N', T, g) représente la fraction $\bar{q} = \phi^C(q)$, où N' est de rang plein selon les lignes.*

Démonstration. Si $N \in \mathbb{Z}^{n \times \ell}$ avec $\text{Rang}(N) = p < n$, alors il existe une matrice $N' \in \mathbb{Z}^{p \times \ell}$ de rang plein selon les lignes et une matrice inversible $C \in \mathbb{Q}^{n \times n}$ telle que $CN = \begin{pmatrix} N' \\ 0 \end{pmatrix}$. D'après la proposition 4.14, la fraction $\bar{q}$ représentée par le triplet (CN, T, g) est égale à $\phi^C(q)$. Comme la matrice CN a $n - p$ lignes nulles, on peut éliminer les $n - p$ dernières variables et le triplet (N', T, g) représente toujours la fraction $\bar{q}$. □

Exemple 4.18. *Soit $q_2 = \frac{2+3abxy}{y^2+5abx} \in \mathbb{K}(U)$ avec $\mathbb{K} = \mathbb{Q}(x, y)$ et $U = \{a, b\}$. Une représentation matricielle de q_2 est*

$$N = \left(\begin{array}{cc|cc} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{array}\right) \begin{matrix} a \\ b \end{matrix}$$

avec $t_1 = 2, t_2 = 3xy, t_3 = y^2, t_4 = 5x$.

En prenant $C = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}$ *et* $N' = \begin{pmatrix} 0 & 1 & 0 & 1 \end{pmatrix}$, *on obtient* $CN = \begin{pmatrix} N' \\ 0 \end{pmatrix}$. *Par conséquent, l'application monomiale* ϕ^C *satisfait* $\phi^C(a) = \bar{a}/\bar{b}$ *et* $\phi^C(b) = \bar{b}$, *donc*

$$\phi^C(q_2) = \frac{2 + 3\bar{a}xy}{y^2 + 5\bar{a}x} \in \mathbb{K}(\bar{U}).$$

Proposition 4.17. *Soit un triplet* (N, T, g) *représentant une fraction* $q \in \mathbb{K}(U)$, *tel que* N *soit de rang plein selon les lignes. Si* $\begin{pmatrix} N \\ \mathbb{1} \end{pmatrix}$ *n'est pas de rang plein selon les lignes, alors il existe une application monomiale* ϕ^C *telle que* (N', T, g) *représente la fraction* $\bar{q} = \phi^C(q)$, *où* $\begin{pmatrix} N' \\ \mathbb{1} \end{pmatrix}$ *est de rang plein selon les lignes.*

Démonstration. Puisque N est de rang plein selon les lignes et puisque $\begin{pmatrix} N \\ \mathbb{1} \end{pmatrix}$ ne l'est pas, on a $\mathbb{1} = \sum_{i=1}^{n} \beta_i N_i$ pour $\beta_i \in \mathbb{Q}$. Sans perte de généralité, on peut supposer que $\beta_n \neq 0$ (quitte à échanger des variables dans l'ensemble U).

En utilisant le corollaire 4.5 et la relation $CN - v\mathbb{1} = \begin{pmatrix} N' \\ 0 \end{pmatrix}$ avec $C = \begin{pmatrix} & & 0 \\ & I & \vdots \\ & & 0 \\ \beta_1 & \cdots & \beta_n \end{pmatrix}$, $v = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}$ et $N' = \begin{pmatrix} N_1 \\ \vdots \\ N_{n-1} \end{pmatrix}$, la fraction $\bar{q}$ représentée par le triplet (CN, T, g) est égale à $\phi^C(q)$. Comme la dernière ligne de CN est nulle, on peut éliminer la dernière variable de U, et le triplet (N', T, g) représente toujours la fraction $\bar{q}$. De plus, la matrice $\begin{pmatrix} N' \\ \mathbb{1} \end{pmatrix}$ est de rang plein selon les lignes. □

Exemple 4.19. *Soit* $q_3 = \frac{2a+3bxy}{ay^2+5bx} \in \mathbb{K}(U)$ *avec* $\mathbb{K} = \mathbb{Q}(x, y)$ *et* $U = \{a, b\}$.

Une représentation matricielle de q_3 *est*

$$N = \left(\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array}\right) \begin{matrix} a \\ b \end{matrix}$$

avec $t_1 = 2, t_2 = 3xy, t_3 = y^2, t_4 = 5x$.

On a $CN - v\mathbb{1} = \begin{pmatrix} N' \\ 0 \end{pmatrix}$ *avec* $C = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, $v = (0, 1)$ *et* $N' = \begin{pmatrix} 1 & 0 & 1 & 0 \end{pmatrix}$. *Par conséquent,* $\bar{q}_3$ *vaut* $\phi^C(q_3) = \frac{2\bar{a}\bar{b}+3\bar{b}xy}{\bar{a}\bar{b}y^2+5\bar{b}x} = \frac{2\bar{a}+3xy}{\bar{a}y^2+5x} \in \mathbb{K}(\bar{a})$. *De plus,* $\bar{q}_3$ *peut être représentée par*

$$N' = \left(\begin{array}{cc|cc} 1 & 0 & 1 & 0 \end{array}\right) \begin{matrix} \bar{a} \end{matrix}$$

où $\begin{pmatrix} N' \\ \mathbb{1} \end{pmatrix}$ *est de rang plein selon les lignes.*

Entrée : Un triplet (N, T, g) représentant une fraction $q \in \mathbb{K}(U)$ telle que $\begin{pmatrix} N \\ \mathbb{1} \end{pmatrix}$ soit de rang plein selon les lignes.

Sortie : Un triplet $(\bar{N}, T, g)$ et une matrice inversible C de $\mathbb{Q}^{n\times n}$, telle que $(\bar{N}, T, g)$ représente $\phi^C(q)$. De plus, $\bar{N}$ est la plus creuse possible et ses coefficients sont dans $\mathbb{Z}$.

1 **Début**

2 Appliquer si nécessaire les propositions 4.16 et 4.17 pour s'assurer que $\begin{pmatrix} N \\ \mathbb{1} \end{pmatrix}$ soit de rang plein selon les lignes ;

3 $\bar{N}, C \leftarrow \mathsf{CSBmodulo}(N, \mathbb{1})$;

4 **Renvoyer** $(\bar{N}, T, g), C$;

Algorithme 11 : $\mathsf{getSparsestFraction}(N, T, g)$

Exemple 4.20. *On applique l'algorithme* getSparsestFraction *sur le triplet* (N, T, g) *de* (4.4)*, représentant la fraction rationnelle* q_1 *de* (4.5)*. La matrice* $\begin{pmatrix} N \\ \mathbb{1} \end{pmatrix}$ *est de rang plein selon les lignes (égal à 3) donc* $\mathsf{getSparsestFraction}(N, T, g)$ *peut appeler* $\mathsf{CSBmodulo}(N, \mathbb{1})$*.*

Pendant l'appel de $\mathsf{CSBmodulo}(N, \mathbb{1})$*, la matrice* $\bar{M}$ *calculée par* $\mathsf{CSB}(M)$ *à la ligne 3, et ensuite triée à la ligne 4 est*

$$\bar{M} = \left(\begin{array}{cc|cc} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{array}\right).$$

La matrice inversible D *calculée à la ligne 5 telle que* $\bar{M} = DM$ *est*

$$D = \begin{pmatrix} 1 & 1 & -1 \\ -2 & -1 & 2 \\ 1 & 0 & 0 \end{pmatrix}.$$

Par ailleurs, la matrice C *calculée par les lignes 6–11 est tout simplement la sous-matrice* 2×2 *de* D *en haut et à gauche, qui est exactement la matrice* (4.10)*.*

Au final, la matrice $\bar{N}$ *calculée par les lignes 6–11 est obtenue en sélectionnant les deux premières lignes de* $\bar{M}$*, nous donnant la matrice* (4.7) *et la fraction* (4.9) *lui correspondant.*

On remarque que si on calculait uniquement CN*, on obtiendrait la représentation matricielle* (4.6) *correspondant à la fraction* (4.8)*, qui n'est pas aussi intéressante que la fraction* (4.9) *representée par* $\bar{N}$*.*

L'exemple suivant montre que l'application monomiale ϕ^C calculée par l'algorithme getSparsestFraction peut faire intervenir des exposants rationnels.

Exemple 4.21. *Soit la représentation matricielle*

$$N = \left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ x & y & x & y \end{array}\right) \begin{matrix} a \\ b \\ c \end{matrix}$$

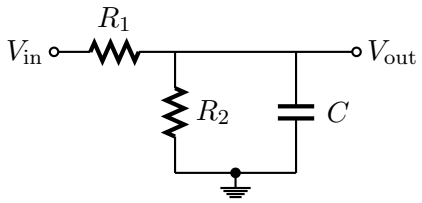

FIGURE 4.1 – Circuit électrique, composé de deux résistances R_1, R_2 et d'un condensateur C.

représentant la fraction $q = \frac{ax+b^3y}{bx+cy} \in \mathbb{K}(a,b,c)$. *Alors l'algorithme* getSparsestFraction(N,T,g) *renvoie*

$$\bar{N} = \left(\begin{array}{cc|cc|c} 1 & 0 & 0 & 0 & \bar{a} \\ 0 & 1 & 0 & 0 & \bar{b} \\ 0 & 0 & 1 & 0 & \bar{c} \\ x & y & x & y & \end{array}\right)$$

avec la matrice $C = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 1/2 \\ -3/2 & -1/2 & -3/2 \end{pmatrix}$.

L'exemple suivant montre que des exposants négatifs sont parfois nécessaires pour les paramètres.

Exemple 4.22. *Soit le triplet suivant*

$$N = \left(\begin{array}{ccc|c} 0 & 1 & -1 & 0 \\ 2 & x & y & z+1 \end{array}\right) \quad a$$

représentant $q = \frac{2+ax+\frac{y}{a}}{z+1} \in \mathbb{K}(a)$ *avec* $\mathbb{K} = \mathbb{Q}(x,y,z)$. *L'algorithme* getSparsestFraction *appliqué au triplet* N *renvoie* N *directement, prouvant que la représentation ci-dessus, qui contient un coefficient négatif, est déjà la plus creuse possible. De plus,* N *est l'unique représentation la plus creuse possible (au sens de ce chapitre), puisque toute addition d'un multiple de* $\mathbb{1}$ *à toute ligne de* N *produit au moins trois nonzéros.*

Exemple 4.23. *La fonction de transfert* $H = V_{\text{out}}/V_{\text{in}}$ *du circuit donné à la figure 4.1 est* $H = \dfrac{R_2}{R_1 + R_2 + iR_1R_2Cw}$, *où* $i^2 = -1$ *et* w *est la fréquence. On considère* H *dans* $\mathbb{K}(R_1, R_2, C)$ *avec* $\mathbb{K} = \mathbb{C}(w)$ *i.e. on considère* R_1, R_2 *et* C *comme des paramètres. On peut montrer que* H *contient une symétrie de type dilatation qui agit sur* R_1, R_2 *et* C. *Par conséquent, en utilisant des techniques basées sur les symétries, on peut éliminer un paramètre par un changement de variables convenable (ce changement de variables n'est pas forcément unique). Par exemple, on peut réécrire* H *sous la forme*

$$H = \frac{x}{1 + x + ix\tau_1 w} \tag{4.11}$$

avec $x = R_2/R_1$ *et* $\tau_1 = R_1C$. *De même, on peut réécrire* H *sous la forme*

$$H = \frac{1}{y + 1 + iy\tau_2 w} \tag{4.12}$$

avec $y = R_1/R_2$ *et* $\tau_2 = R_2C$. *D'autres changements de variables sont également possibles mais ne sont pas donnés ici.*

Les relations (4.11) *et* (4.12) *ne sont pas les plus creuses. La relation* (4.12) *peut facilement être rendue plus creuse avec le changement de variables* $a = y\tau_2$*, ce qui donne* $H = \frac{1}{y+1+iaw}$.

Cependant, la relation (4.11) *nécessite un traitement légèrement plus subtile, en divisant d'abord simultanément le numérateur et le dénominateur par* x *donc en écrivant* $H = \frac{1}{\frac{1}{x}+1+i\tau_1 w}$*, puis en posant* $y = 1/x$ *(on remarque que* $a = \tau_1$ *puisque* $a = y\tau_2 = (R_1/R_2)R_2C = R_1C = \tau_1$*). La division par* x *sera automatiquement trouvée par* getSparsestFraction *appliqué au triplet représentant* (4.11)

$$N = \left(\begin{array}{c|ccc} 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & iw \end{array}\right) \begin{array}{c} x \\ \tau_1 \\ \end{array}$$

grâce à la ligne $\mathbb{1}$ *ajoutée à* N *durant l'appel de* CSB*, qui permet de remplacer la première ligne* N_1 *de* N *par* $N_1 - \mathbb{1} = \begin{pmatrix} 0 & | & -1 & 0 & 0 \end{pmatrix}$*. De plus, notre implantation de* CSB *transformera la ligne* $\begin{pmatrix} 0 & | & -1 & 0 & 0 \end{pmatrix}$ *en son opposé puisqu'elle ne contient que des coefficient négatifs.*

4.3 Sommes de fractions rationnelles

Nous considérons désormais une somme de fractions rationnelles de $\mathbb{K}(U)$. On pourrait la transformer en une seule fraction en réduisant les fractions de la somme au même dénominateur. Cependant, nous souhaitons éviter une telle manipulation pour deux raisons. Premièrement, cela peut accroître considérablement la taille du numérateur ainsi que celle du dénominateur. Deuxièmement, un utilisateur pourrait souhaiter conserver une expression sous la forme d'une somme de fractions. Ce dernier point apparaît par exemple dans le contexte biologique avec des expressions de la forme $p+\sum_i \frac{V_i x_i}{x_i+k_i}$ où les x_i sont des concentrations, p est un polynôme en les x_i et d'autres paramètres, tandis que les V_i et k_i sont des constantes de termes de Michaelis-Menten [31, 46].

4.3.1 Représentation matricielle d'une somme de fractions

Définition 4.24. *Soient* s *fractions* $q_i \in \mathbb{K}(U)$*, où chaque fraction* q_i *est représentée par un triplet* (N^i, T^i, g^i)*. Alors, l'ensemble des triplets* $H = \{(N^i, T^i, g^i)_{1\leq i\leq s}\}$ *est appelé* représentation matricielle de la somme $S = \sum_{i=1}^{s} q_i$*. L'ensemble* H *des triplets peut être représenté par la matrice suivante, où les doubles barres séparent les représentations matricielles de deux fractions distinctes :*

$$N = \begin{pmatrix} N^1 \parallel N^2 \parallel \ldots \parallel N^s \end{pmatrix}.$$

Exemple 4.24. *Soit la somme de fractions rationnelles* S_1 *définie par*

$$S_1 = \frac{2a + abxy}{ay^2 + 7bx} + abcy + \frac{abcy^2 + 3ay}{ax}.$$

Une représentation matricielle possible de S_1 *est*

$$\begin{array}{c} \left(\begin{array}{cc|cc||c|c||cc|c} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{array}\right) \begin{array}{c} a \\ b \\ c \end{array} \\ \begin{array}{ccccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 \end{array} \end{array} \tag{4.13}$$

with $t_1 = 2, t_2 = xy, t_3 = y^2, t_4 = 7x, t_5 = y, t_6 = 1, t_7 = y^2, t_8 = 3y, t_9 = x$.

Soit une représentation matricielle $(N^i, T^i, g^i)_{1\leq i\leq s}$ d'une somme $S = \sum q_i$. Comme la proposition 4.13 peut être appliquée indépendamment sur chaque fraction q_i, on peut introduire la proposition 4.18 qui généralise la proposition 4.13, après avoir introduit la matrice $\mathbb{1}^S$ de taille $s \times (\sum_{i=1}^{s} \mathrm{card}(T^i))$, diagonale par blocs, définie par

$$\mathbb{1}^S = \begin{pmatrix} \mathbb{1}_{\mathrm{card}(T_1)} & & & \\ & \mathbb{1}_{\mathrm{card}(T_2)} & & \\ & & \ddots & \\ & & & \mathbb{1}_{\mathrm{card}(T_s)} \end{pmatrix}. \tag{4.14}$$

Proposition 4.18. *Soit un ensemble de triplets (N^i, T^i, g^i) représentant une somme de fractions rationnelles $S = \sum_{i=1}^{s} q_i$ écrite comme à la définition 4.24. Pour tout $V \in \mathbb{Z}^{n\times s}$, $\bar{N} = N + V\mathbb{1}^S$ est une représentation matricielle de S.*

Démonstration. Appliquer la proposition 4.13 sur chaque fraction q_i. □

4.3.2 Action d'une application monomiale

La proposition 4.19 et le corollaire 4.6 suivants sont respectivement les généralisations de la proposition 4.14 et du corollaire 4.5 pour des sommes de fractions.

Proposition 4.19. *Soit un ensemble de triplets (N^i, T^i, g^i) représentant une somme de fractions rationnelles $S = \sum_{i=1}^{s} q_i$ écrit comme à la définition 4.24. Soit une matrice inversible C de $\mathbb{Q}^{n\times n}$. Si CN contient uniquement des coefficients dans $\mathbb{Z}$, alors CN est une représentation matricielle de $\bar{S} = \phi^C(S)$.*

Corollaire 4.6. *Soit un ensemble de triplets (N^i, T^i, g^i) représentant une somme de fractions rationnelles $S = \sum_{i=1}^{s} q_i$ écrit comme à la définition 4.24. Soient une matrice inversible C de $\mathbb{Q}^{n\times n}$ et une matrice V de $\mathbb{Q}^{n\times s}$. Si $CN + V\mathbb{1}^S$ contient uniquement des coefficients entiers, alors $CN + V\mathbb{1}^S$ est une représentation matricielle de $\bar{S} = \phi^C(S)$.*

Exemple 4.25. *Soit l'application $\phi(a) = \bar{a}$, $\phi(b) = \bar{a}/\bar{b}$, $\phi(c) = \bar{b}\bar{c}/\bar{a}$ définie par la matrice inversible C suivante :*

$$\begin{pmatrix} 1 & 1 & -1 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \\ a & b & c \end{pmatrix} \begin{matrix} \bar{a} \\ \bar{b} \\ \bar{c} \end{matrix}. \tag{4.15}$$

Avec ce changement de variables, la fraction S_1 de l'exemple 4.24 devient

$$\bar{S}_1 = \frac{2\bar{a} + \frac{\bar{a}^2}{\bar{b}}xy}{\bar{a}y^2 + 7\frac{\bar{a}}{\bar{b}}x} + \bar{a}\bar{c}y + \frac{\bar{a}\bar{c}y^2 + 3\bar{a}y}{\bar{a}x}. \tag{4.16}$$

Soit N la représentation matricielle (4.13) de S_1, la matrice CN est une représentation matricielle de $\bar{S}_1 = \phi^C(S_1)$. La matrice CN peut néanmoins être rendue plus creuse en considérant

$$V = \begin{pmatrix} -1 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

et en appliquant le corollaire 4.6. En effet, la matrice suivante $\bar{N} = CN + V\mathbb{1}^{S_1}$ *est également une représentation matricielle de* $\bar{S}_1$ *:*

$$\left(\begin{array}{cc|cc||c|c||cc|c} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{array}\right) \begin{array}{c} \bar{a} \\ \bar{b} \\ \bar{c} \end{array}, \tag{4.17}$$

et elle représente

$$\bar{S}_1 = \frac{2\bar{b} + \bar{a}xy}{\bar{b}y^2 + 7x} + \bar{a}\bar{c}y + \frac{\bar{c}y^2 + 3y}{x}. \tag{4.18}$$

On remarque qu'ajouter $V\mathbb{1}^{S_1}$ *à* CN *revient à multiplier par* $\frac{\bar{b}}{\bar{a}}$ *le numérateur et le dénominateur de la première fraction, et diviser par* $\bar{a}$ *le numérateur et le dénominateur de la troisième fraction dans* (4.16).

4.3.3 Algorithme getSparsestSumOfFractions

L'algorithme getSparsestSumOfFractions repose sur les mêmes idées que celles présentées dans la section 4.2 : étant donné un ensemble de triplets (N^i, T^i, g^i) représentant une somme de fractions rationnelles $S = \sum_{i=1}^{s} q_i$ écrit comme à la définition 4.24, on peut rechercher une matrice inversible C et une matrice V telles que $CN + V\mathbb{1}^S$ soit la plus creuse possible.

Nous présentons d'abord la proposition 4.20 ci-dessous, qui est une légère généralisation de la proposition 4.17. Comme dans la section 4.2, la proposition 4.20 est nécessaire lorsque la somme de fractions contient des symétries de type dilation en les variables de U.

Proposition 4.20. *Soit un ensemble de triplets* (N^i, T^i, g^i) *représentant une somme de fractions rationnelles* $S = \sum_{i=1}^{s} q_i$ *comme écrit à la définition 4.24. On suppose que* N *est de rang plein selon les lignes.*

Si $\begin{pmatrix} N \\ \mathbb{1}^S \end{pmatrix}$ *n'est pas de rang plein selon les lignes, alors il existe une application monomiale* ϕ^C *ainsi que des matrices* N'^i *telles que l'ensemble des triplets* (N'^i, T^i, g^i) *représente la fraction* $\bar{S} = \phi^C(S)$*, où* $\begin{pmatrix} N' \\ \mathbb{1} \end{pmatrix}$ *est de rang plein selon les lignes.*

Démonstration. On suppose que la matrice $M = \begin{pmatrix} N \\ \mathbb{1}^S \end{pmatrix}$ n'est pas de rang plein selon les lignes. Alors il existe une dépendance linéaire non triviale entre les lignes de M. Comme les matrices N et $\mathbb{1}^S$ sont toutes les deux de rang plein selon les lignes, la dépendance linéaire implique l'existence d'une ligne de N avec un coefficient non nul. La fin de la preuve est similaire à celle de la proposition 4.17. □

Exemple 4.26. *On considère de nouveau, à partir de l'exemple 4.24, la fraction* S_1 *ainsi que sa représentation matricielle* N *donnée par l'équation* (4.13)*. On vérifie que la matrice* $\begin{pmatrix} N \\ \mathbb{1}^S \end{pmatrix}$ *est bien de rang plein selon les lignes. Alors, on applique* getSparsestSumOfFractions(N)*, qui appelle* CSBmodulo$(N, \mathbb{1}^S)$*.*

Notre implantation de CSBmodulo *renvoie la matrice* $\bar{N}$ *décrite par l'équation* (4.17) *et la matrice* C *décrite par l'équation* (4.15)*. Enfin, nous concluons qu'une représentation simplifiée* $\bar{S}_1 = \phi(S_1)$ *peut s'écrire sous la forme donnée par l'équation* (4.18)*.*

Entrée : Un ensemble $H = \{(N^i, T^i, g^i)_{1\leq i\leq s}\}$ représentant une somme de fractions rationnelles $S = \sum_{i=1}^{s} q_i \in \mathbb{K}(U)$ écrit comme à la définition 4.24.
Sortie : Un ensemble $\bar{H} = \{(\bar{N}^i, T^i, g^i)_{1\leq i\leq s}\}$ et une matrice inversible C de $\mathbb{Q}^{n\times n}$, tels que $\bar{H}$ représente $\phi^C(q)$. De plus, $\bar{N} = \left(\bar{N}^1||\cdots||\bar{N}^s\right)$ est la plus creuse possible et contient uniquement des coefficients dans $\mathbb{Z}$.

Début
  Appliquer si nécessaire les propositions 4.16 et 4.20 pour s'assurer que $\begin{pmatrix} N \\ \mathbb{1}^S \end{pmatrix}$ soit de rang plein selon les lignes ;
  $\bar{N}, C \leftarrow \mathsf{CSBmodulo}(N, \mathbb{1}^S)$;
  Écrire $\bar{N}$ sous la forme $\left(\bar{N}^1||\cdots||\bar{N}^s\right)$;
  Renvoyer $\{(\bar{N}^i, T^i, g^i)_{1\leq i\leq s}\}, C$;

Algorithme 12 : $\mathsf{getSparsestSumOfFractions}(H)$

4.3.4 Application à la simplification de systèmes d'équations

Les techniques présentées pour les sommes de fractions rationnelles peuvent être adaptées pour des systèmes d'équations différentielles de la forme $X'(t) = F(\Theta, X(t))$ (où $X(t)$ est un vecteur de fonctions, $F(\Theta, X(t))$ est un vecteur de fractions et les Θ sont des paramètres), comme dans le système (4.1). En effet, on peut appliquer l'algorithme getSparsestSumOfFractions à la somme de fractions $\sum_{i=1}^{s} F_i(\Theta, X(t))$, dans laquelle les $F_i(\Theta, X(t))$ désignent les composants du vecteur $F(\Theta, X(t))$.

Exemple 4.27. *On considère à nouveau le système* (4.1) *:*

$$\begin{cases} G(t)' = \theta(1-G(t)) - \alpha k_1 k_2 k_3 P(t)^4 G(t) \\ M(t)' = \rho_b(1-G(t)) + \rho_f G(t) - \delta_M M(t) \\ P(t)' = \dfrac{4\theta(1-G(t)) - 4\alpha k_1 k_2 k_3 P(t)^4 G(t) - \delta_P P(t) + \beta M(t)}{16 k_1 k_2 k_3 P(t)^3 + 9 k_1 k_2 P(t)^2 + 4 k_1 P(t) + 1}. \end{cases}$$

Soit la somme S_2 des trois membres droits considérés comme des fractions de $\mathbb{K}(k_1, k_2, k_3)$, où $\mathbb{K} = \mathbb{Q}(G(t), M(t), P(t), \theta, \alpha, \beta, \rho_b, \rho_f, \delta_M, \delta_P)$. Une représentation matricielle $N = \left(N^1||N^2||N^3\right)$ de S_2 est donnée par

$$N^1 = \left(\begin{array}{cc|c} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ t_1 & t_2 & t_3 \end{array}\right) \begin{matrix} k_1 \\ k_2 \\ k_3 \end{matrix},$$

$$N^2 = \left(\begin{array}{c|c} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ t_4 & t_5 \end{array}\right) \begin{matrix} k_1 \\ k_2 \\ k_3 \end{matrix},$$

$$N^3 = \left(\begin{array}{cc|cccc} 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ t_6 & t_7 & t_8 & t_9 & t_{10} & t_{11} \end{array}\right) \begin{matrix} k_1 \\ k_2 \\ k_3 \end{matrix}$$

avec $t_1 = \theta(1-G(t))$, $t_2 = -\alpha P(t)^4 G(t)$, $t_3 = 1$, $t_4 = \rho_b(1-G(t)) + \rho_f G(t) - \delta_M M(t)$, $t_5 = 1$, $t_6 = 4\theta(1-G(t)) - \delta_P P(t) + \beta M(t)$, $t_7 = -4\alpha P(t)^4 G(t)$, $t_8 = 16P(t)^3$, $t_9 = 9P(t)^2$, $t_{10} = 4P(t)$, $t_{11} = 1$.

L'algorithme getSparsestSumOfFractions *produit le système simplifié* (4.2) *rappelé ici :*

$$\begin{cases} G(t)' = \theta(1-G(t)) - \alpha\bar{k}_3 P(t)^4 G(t) \\ M(t)' = \rho_b(1-G(t)) + \rho_f G(t) - \delta_M M(t) \\ P(t)' = \dfrac{4\theta(1-G(t)) - 4\alpha\bar{k}_3 P(t)^4 G(t) - \delta_P P(t) + \beta M(t)}{16\bar{k}_3 P(t)^3 + 9\bar{k}_2 P(t)^2 + 4\bar{k}_1 P(t) + 1} \end{cases}$$

ansi que l'application monomiale ϕ *définie par*

$$\phi(k_1) = \bar{k}_1, \quad \phi(k_2) = \bar{k}_2/\bar{k}_1, \quad \phi(k_3) = \bar{k}_3/\bar{k}_2$$

et codée par la matrice $C = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix}$. *Le lecteur remarquera que sur cet exemple, la matrice* $\mathbb{1}^S$ *n'a pas été utilisée dans les calculs, puisque le système simplifié* (4.2) *correspond à* $\bar{N} = CN$.

Chaque membre droit du système (4.1) est une fraction rationnelle. On peut également appliquer la technique précédente dans le cas où les membres droits sont des sommes de fractions. Conceptuellement, on pourrait encore étendre la définition 4.21 afin de représenter un système de p équations avec une représentation matricielle sous la forme $N = \left(N^1 ||| N^2 ||| \cdots ||| N^p\right)$ où chaque N^i est une représentation matricielle de chaque équation sous la forme d'une somme de fractions rationnelles. On sépare par 3 barres ces représentations matricielles.

Il n'est pas nécessaire de redéfinir les concepts introduits pour les sommes de fractions aux systèmes, étant donné qu'ils n'impliquent aucun raisonnement particulier à généraliser par rapport à ce qui a été réalisé dans cette section.

4.3.5 Complexité et implantation

Excepté l'appel de l'algorithme CSBmodulo, les instructions de l'algorithme getSparsestFraction sont réalisées au plus en $O(\ell^3)$. La complexité de getSparsestFraction est dominée par celle de CSBmodulo, qui a la même complexité que CSB.

Proposition 4.21. *Avec les notations de la définition 4.21,* getSparsestFraction(N, T, g) *est de complexité* $O(\mathcal{N}(N)\ell^3 2^\ell)$.

Remarque 4.8. *À caise de la présence du vecteur ligne* $\mathbb{1}$ *présent dans l'algorithme* getSparsestFraction*, on ne tire malheureusement pas partie de la situation "d est très petit devant n" (voir section 2.4.1).*

L'implantation des algorithmes CSBmodulo, getSparsestFraction et getSparsestSumOfFractions a été réalisée sous Maple. Nous avons également implanté une fonction qui prend en entrée un système d'équations représenté par une liste de sommes de fractions rationnelles et renvoie en sortie un système d'équations simplifié dans le sens de l'algorithme getSparsestSumOfFractions, comme expliqué à la fin de la section 4.3.4.

Chapitre 5

Reverse Engineering de modèles

Sommaire

5.1 Introduction 75
5.2 Modèle BIOMD253 76
5.2.1 Mise en place d'un programme linéaire 77
5.2.2 Expérimentations sur différents objectifs 78
5.3 Modèle BIOMD050 79

Ce chapitre a été motivé par l'impossibilité, dans certains cas, de concilier caractère creux et positivité des bases de lois de conservation. Nous avons vu dans la section 1.2.6 que les lois de conservation à coefficients positifs ne généraient pas toujours l'ensemble des lois de conservation. Dans de tels cas, toute base complète de lois de conservation contient nécessairement au moins une loi contenant à la fois des coefficients positifs et négatifs. D'après la remarque 1.3, il y a 14 modèles réarrangés de BioModels pour lesquels ce phénomène se produit. Nous nous sommes posés la question naturelle suivante : "peut-on modifier le système de réactions chimiques afin que l'ensemble des lois de conservation soit généré uniquement par des lois de conservation à coefficients positifs ?" Pour y parvenir, nous avons cherché à faire apparaître de nouvelles lois de conservation à coefficients positifs, en introduisant une ou plusieurs nouvelles espèces dans le système de réactions. Il y a certainement d'autres façons d'y parvenir, notamment en supprimant/déplaçant des espèces dans les réactions, voire en supprimant des réactions. Nous n'avons pas testé ces approches, mais cela pourrait être réalisé dans un travail futur.

Ce chapitre présente un schéma d'algorithme permettant de suggérer des modèles enrichis d'une nouvelle espèce chimique, afin d'augmenter la dimension de l'espace des lois de conservation à coefficients positifs. Nous illustrons notre schéma d'algorithme sur les modèles BIOMD253 et BIOMD050. Dans un travail ultérieur, nous nous intéresserons aux $14-2=12$ modèles restants.

5.1 Introduction

L'ajout d'espèces nous a semblé naturel sur le modèle minimaliste de l'exemple 1.9

$$A + B \rightharpoonup \emptyset.$$

Ce système admet pour unique loi de conservation $A - B = constante$. Si on ajoute une espèce N comme produit afin d'obtenir

$$A + B \rightharpoonup N,$$

on obtient alors deux lois de conservation à coefficients uniquement positifs : $A+N = constante$ et $B+N = constante$. Le modèle ainsi obtenu après ajout de l'espèce N possède donc un espace de lois de conservation de dimension 2, généré par deux lois à coefficients positifs.

On remarque au passage que la loi de conservation initiale $A - B = constante$ (qui reste d'ailleurs valide dans le modèle modifié) est simplement la différence entre les deux lois $A+N = constante$ et $B + N = constante$. Ainsi, la loi $A - B = constante$ laisse espérer l'existence de deux lois de conservation positives, l'une de la forme $A + \cdots = constante$, l'autre de la forme $B + \cdots = constante$, une fois qu'une nouvelle espèce N est correctement ajoutée dans les réactions.

Cela suggère le schéma d'algorithme qui sera détaillé dans les sections suivantes sur les modèles réarrangés BIOMD253 et BIOMD050 de BioModels :

1. considérer une loi de conservation qui n'est pas générée par les P-invariants, représentée par un vecteur y ;
2. écrire y sous la forme $y = p - q$ où (p, q) est le représentant positif de y (définition 3.15) ;
3. introduire une nouvelle espèce N dans les réactions (de préférence dans un minimum d'entre elles) afin qu'il existe un vecteur x positif et un rationnel $k \in \mathbb{Q}_+^*$ tels que $\binom{p+x}{k}$ et $\binom{q+x}{k}$ représentent deux lois de conservation positives dans le modèle enrichi de cette nouvelle espèce N. Cette étape peut se résoudre par la programmation linéaire.

En suivant ce schéma d'algorithme sur le modèle SBML BIOMD253, nous avons obtenu différents modèles enrichis d'une nouvelle espèce ne contenant que des lois de conservation à coefficients positifs. Après coup, nous avons constaté que l'un de nos modèles enrichis était en fait le modèle de l'article [58], et que le modèle SBML BIOMD253 avait été écrit à partir de [58] en supprimant une espèce.

La section 5.2 présente une étude du modèle SBML BIOMD253, ainsi que le schéma d'algorithme basé sur la programmation linéaire. Pour finir, la section 5.3 présente une suggestion de deux modèles pour le modèle SBML BIOMD050 tiré de [44].

5.2 Modèle BIOMD253

Rappelons le modèle BIOMD253 [58] obtenu à partir du modèle SBML de BioModels [1] :

$$\mathcal{S} : \begin{cases} Glc + ATP & \rightharpoonup & HMP \\ HMP + ATP & \rightharpoonup & Fru16P2 \\ Fru16P2 + 4\ ADP & \rightharpoonup & 4\ ATP \\ ATP & \rightharpoonup & ADP. \end{cases} \tag{5.1}$$

Si l'on oublie l'espèce $Tre6P$ qui est présente dans le modèle mais pas dans les réactions, la seule loi de conservation est

$$-HMP + ATP - 2\ Glc + ADP = constante, \tag{5.2}$$

qui contient des coefficients positifs et négatifs, que l'on ne peut transformer en coefficients uniquement positifs. Toutefois, l'article décrivant le modèle BIOMD253 [58] fait intervenir l'espèce

$EtOH$ qui a été supprimée lors de l'écriture du modèle SBML :

$$\bar{\mathcal{S}} : \begin{cases} Glc + ATP & \rightharpoonup & HMP \\ HMP + ATP & \rightharpoonup & Fru16P2 \\ Fru16P2 + 4\ ADP & \rightharpoonup & \mathbf{2\ EtOH} + 4\ ATP \\ ATP & \rightharpoonup & ADP. \end{cases} \tag{5.3}$$

qui admet les lois de conservation positives

$$ATP + ADP + HMP + 2\ Fru16P2 + EtOH = constante, \tag{5.4}$$

$$2\ HMP + 2\ Glc + 2\ Fru16P2 + EtOH = constante. \tag{5.5}$$

On remarque au passage que la loi (5.2) est égale à la différence entre la loi (5.4) et la loi (5.5).

Comme la loi de conservation (5.2) contient des coefficients positifs et négatifs, nous suivons le schéma d'algorithme évoqué dans la section 5.1 afin d'obtenir deux nouvelles lois de conservation positives, grâce à l'ajout d'une nouvelle espèce N dans le modèle. Nous choisissons de classer les espèces dans l'ordre : HMP, $Fru16P2$, ATP, Glc, ADP. Soit $y = {}^t\begin{pmatrix}-1 & 0 & 1 & -2 & 1\end{pmatrix}$ le vecteur de dimension 5 représentant la loi de conservation (5.2). Son représentant positif (p, q) est défini par $p = {}^t\begin{pmatrix}0 & 0 & 1 & 0 & 1\end{pmatrix}$ et $q = {}^t\begin{pmatrix}1 & 0 & 0 & 2 & 0\end{pmatrix}$. On recherche un vecteur x positif de dimension 5 et un rationnel $k \in \mathbb{Q}_+^*$, tels que $\begin{pmatrix} p+x \\ k \end{pmatrix}$ et $\begin{pmatrix} q+x \\ k \end{pmatrix}$ représentent deux lois de conservation positives. Soit M la matrice de stœchiométrie du modèle initial (de dimension 5×4). Ajouter une espèce N dans ce modèle dont les coefficients stœchiométriques sont inconnus revient à considérer la matrice $\begin{pmatrix} M \\ {}^t w \end{pmatrix}$ où w est un vecteur de $\mathbb{Q}^4$. Les variables x, w et k sont inconnues. Nous souhaitons les déterminer. Si on arrive à déterminer ces variables, la suggestion de modèle est alors contenue dans le vecteur w qui contient le coefficient stœchiométrique de la nouvelle espèce N dans chaque réaction.

5.2.1 Mise en place d'un programme linéaire

Avec les notations précédentes, on recherche x, k et w satisfaisant

$$\begin{pmatrix} {}^t M & w \end{pmatrix} \begin{pmatrix} p+x \\ k \end{pmatrix} = 0$$

qui s'écrit de façon équivalente

$${}^t M x + kw = -{}^t M p. \tag{5.6}$$

Le système (5.6) est non linéaire à cause du terme kw. Par chance, le système (5.6) admet une symétrie de dilatation agissant sur k et w : pour toute solution (x^0, k^0, w^0), alors $(x^0, \lambda k^0, w^0/\lambda)$ est aussi solution pour tout $\lambda > 0$. Intuitivement, cette symétrie traduit le fait qu'il y a un degré de liberté sur les coefficients stœchiométriques de l'espèce à rajouter. Comme k est non nul (car $k \in \mathbb{Q}_+^*$), on peut supposer sans perte de généralité, que $k = 1$. Grâce à cette hypothèse, le système (5.6) devient linéaire :

$${}^t M x + w = -{}^t M p. \tag{5.7}$$

Comme il y a des conditions de signe sur x à respecter, nous sommes amenés à considérer le programme linéaire suivant, dans lequel un objectif dépendant de x et w doit être choisi :

$$\mathcal{P}(M, p, f, g) : \begin{cases} {}^t M x + w = -{}^t M p \\ z = f(w) + g(x)[\min] \\ x \in \mathbb{Q}_+^n, w \in \mathbb{Q}^m \end{cases} \tag{5.8}$$

où les inconnues sont x et w.

Le programme linéaire (5.8) admet toujours des solutions réalisables, car il suffit de choisir $x = 0$ et $w = -{}^tMp$.

Il est naturel de vouloir perturber au minimum le système de réactions en faisant intervenir le moins possible la nouvelle espèce. Mathématiquement, cela revient à chercher un vecteur w creux. En s'inspirant des techniques du chapitre 3, on peut par exemple minimiser des objectifs

$$z = \sum_{i \neq j} |w_i|$$

qui ont pour effet de minimiser en valeur les coefficients w_i tels que $i \neq j$, où j est un entier fixé. Cela revient à chercher une suggestion de modèle où la nouvelle espèce apparaît en priorité dans la réaction j et le moins possible dans les autres réactions.

5.2.2 Expérimentations sur différents objectifs

Pour des raisons de clarté, la nouvelle espèce N est appelée $EtOH$ dans cette section.

L'objectif $z = |w_2| + |w_3| + |w_4|$

Cet objectif minimise la présence de la nouvelle espèce $EtOH$ dans les réactions 2, 3, et 4. Il fournit la solution $x = {}^t\begin{pmatrix}0 & 4/3 & 1/3 & 0 & 0\end{pmatrix}$ et $w = {}^t\begin{pmatrix}4/3 & 0 & 0 & 1/3\end{pmatrix}$ ce qui correspond au système

$$\begin{cases} Glc + ATP & \rightharpoonup HMP \\ HMP + ATP & \rightharpoonup Fru16P2 + 4/3\ EtOH \\ Fru16P2 + 4\ ADP & \rightharpoonup 4\ ATP + 1/3\ EtOH \\ ATP & \rightharpoonup ADP \end{cases}$$

qui admet les lois de conservation positives

$$\begin{aligned} 4/3\ Fru16P2 + 4/3\ ATP + ADP + EtOH &= constante \\ HMP + 4/3\ Fru16P2 + 1/3\ ATP + 2\ Glc + EtOH &= constante. \end{aligned}$$

L'objectif $z = |w_1| + |w_3| + |w_4|$

Cet objectif minimise la présence de la nouvelle espèce $EtOH$ dans les réactions 1, 3, et 4. Il fournit la solution $x = {}^t\begin{pmatrix}1 & 0 & 0 & 0 & 0\end{pmatrix}$ et $w = {}^t\begin{pmatrix}0 & 2 & 0 & 0\end{pmatrix}$ ce qui correspond au système

$$\begin{cases} Glc + ATP & \rightharpoonup HMP \\ HMP + ATP & \rightharpoonup Fru16P2 + 2\ EtOH \\ Fru16P2 + 4\ ADP & \rightharpoonup 4\ ATP \\ ATP & \rightharpoonup ADP \end{cases}$$

qui admet les lois de conservation positives

$$\begin{aligned} HMP + ATP + ADP + EtOH &= constante \\ 2\ HMP + 2\ Glc + EtOH &= constante. \end{aligned}$$

L'objectif $z = |w_1| + |w_2| + |w_4|$

Cet objectif minimise la présence de la nouvelle espèce $EtOH$ dans les réactions 1, 2, et 4. Il fournit la solution $x = {}^t\begin{pmatrix}1 & 2 & 0 & 0 & 0\end{pmatrix}$ et $w = {}^t\begin{pmatrix}0 & 0 & 2 & 0\end{pmatrix}$ qui correspond exactement au système de réactions (5.3), à savoir l'ajout de 2 $EtOH$ dans les produits de la réaction 3.

De plus, les vecteurs $\begin{pmatrix}p+x\\1\end{pmatrix}$ et $\begin{pmatrix}q+x\\1\end{pmatrix}$ sont exactement les lois de conservation positives (5.4) et (5.5).

Remarquons que c'est par hasard que nous avons trouvé le bon coefficient stœchiométrique égal à 2 dans la réaction 3. Cela provient du fait que le coefficient de $EtOH$ dans les lois (5.4) et (5.5) est égal à 1, ce qui correspond à notre hypothèse de fixer $k = 1$.

L'objectif $z = |w_1| + |w_2| + |w_3|$

Cet objectif minimise la présence de la nouvelle espèce $EtOH$ dans les réactions 1, 2, et 3. Il fournit la solution $x = {}^t\begin{pmatrix}2 & 4 & 1 & 0 & 0\end{pmatrix}$ et $w = {}^t\begin{pmatrix}0 & 0 & 0 & 1\end{pmatrix}$ ce qui correspond au système

$$\begin{cases} Glc + ATP & \rightharpoonup HMP \\ HMP + ATP & \rightharpoonup Fru16P2 \\ Fru16P2 + 4\ ADP & \rightharpoonup 4\ ATP \\ ATP & \rightharpoonup ADP + EtOH \end{cases}$$

qui admet les lois de conservation positives

$$\begin{aligned} 2\ HMP + 4\ Fru16P2 + 2\ ATP + ADP + EtOH &= constante \\ 3\ HMP + 4\ Fru16P2 + ATP + 2\ Glc + EtOH &= constante. \end{aligned}$$

L'objectif $z = \sum_{i=1}^{5}(x_i + p_i)$

Cet objectif est différent des quatre précédents. Il permet de privilégier des nouvelles lois de conservation de faible norme, qui on peut l'espérer, sont également des lois creuses. Il fournit la solution $x = 0$ $w = {}^t\begin{pmatrix}1 & 1 & 0 & 0\end{pmatrix}$ ce qui correspond au système

$$\begin{cases} Glc + ATP & \rightharpoonup HMP + EtOH \\ HMP + ATP & \rightharpoonup Fru16P2 + EtOH \\ Fru16P2 + 4\ ADP & \rightharpoonup 4\ ATP \\ ATP & \rightharpoonup ADP \end{cases}$$

dont les lois de conservation sont :

$$\begin{aligned} ATP + ADP + EtOH &= constante \\ HMP + 2\ Glc + EtOH &= constante. \end{aligned}$$

5.3 Modèle BIOMD050

Dans [44], on dispose d'un modèle initial ("Scheme 1") qui est simplifié en un deuxième modèle ("Scheme 2"), contenant en particulier les réactions r_3 : $DFG \rightharpoonup MG + Gly$ et r_{15} : $MG + C_n \rightharpoonup FA + AA$. D'après [44], ce deuxième modèle n'est pas correct d'un point de vue chimique car l'hypothèse selon laquelle MG se forme directement à partir de DFG s'avère fausse. Les auteurs pensent qu'il existe une espèce intermédiaire entre DFG et MG. Comme

ils supposent qu'il s'agit de l'espèce C_n, ils formulent un troisième modèle ("Scheme 3") où ils considèrent les mêmes réactions que dans le deuxième modèle, mais déplacent MG dans le membre droit de la réaction r_{15} et enlèvent MG de la réaction r_3. Ce troisième modèle correspond au modèle SBML BIOMD050 de BioModels [1], dont le système de réactions

$$\mathcal{S} : \begin{cases} r_1 : & DFG & \rightharpoonup & E_1 \\ r_2 : & DFG & \rightharpoonup & E_2 \\ r_3 : & DFG & \rightharpoonup & C_n + Gly \\ r_4 : & E_1 & \rightharpoonup & Gly + 3\_DG \\ r_5 : & 3\_DG & \rightharpoonup & C_n \\ r_6 : & 3\_DG & \rightharpoonup & FA \\ r_7 : & E_2 & \rightharpoonup & Gly + 1\_DG \\ r_8 : & 1\_DG & \rightharpoonup & C_n \\ r_9 : & 1\_DG & \rightharpoonup & AA \\ r_{10} : & E_1 & \rightharpoonup & Gly + Man \\ r_{11} : & E_1 & \rightharpoonup & Gly + Glu \\ r_{12} : & Man & \rightharpoonup & Glu \\ r_{13} : & Glu & \rightharpoonup & 3\_DG \\ r_{14} : & C_n + Gly & \rightharpoonup & Mel \\ r_{15} : & C_n & \rightharpoonup & FA + AA + MG \\ r_{16} : & E_2 & \rightharpoonup & Gly + Fru \end{cases} \tag{5.9}$$

admet les lois de conservation (formant une base la plus creuse)

$$DFG + E_1 + E_2 + Gly + Mel = constante, \tag{5.10}$$

$$C_n - Gly + 3\_DG + FA + 1\_DG + AA + Man + Glu - MG + Fru = constante. \tag{5.11}$$

La loi de conservation (5.11) contient des coefficients positifs et négatifs, que l'on ne peut transformer en coefficients uniquement positifs par combinaison linéaire avec la loi (5.10).

Nous suivons à nouveau le schéma d'algorithme évoqué dans la section 5.2.1 en résolvant le système (5.8).

En testant les 16 objectifs de la forme $z = \sum_{i \neq j} |w_i|$ en faisant varier j de 1 à 16, on obtient uniquement les deux suggestions de systèmes ci-dessous (la suggestion 1 est fournie quinze fois, la suggestion 2 une seule fois).

Suggestion 1

La réaction r_{15} est modifiée en

$$r_{15} : C_n + N \rightharpoonup MG + FA + AA. \tag{5.12}$$

qui a pour loi de conservation (après simplification par CSB)

$$\begin{aligned} DFG + E_1 + E_2 + Gly + Mel &= constante, \\ MG + N &= constante, \\ DFG + E_1 + E_2 + Mel + C_n + 3\_DG + FA + 1\_DG + AA + Man + Glu + Fru + N &= constante. \end{aligned}$$

Suggestion 2

Les deux réactions r_{14} et r_{15} sont modifiées :

$$r_{14} \quad : \quad C_n + Gly \rightharpoonup Mel + N \tag{5.13}$$
$$r_{15} \quad : \quad C_n + N \rightharpoonup MG + FA + AA. \tag{5.14}$$

et les nouvelles lois de conservation sont

$$\begin{aligned} DFG + E_1 + E_2 + Gly + Mel &= constante, \\ MG + N &= constante, \\ DFG + E_1 + E_2 + C_n + 3\_DG + FA + 1\_DG & \\ + AA + Man + Glu + Fru + N &= constante. \end{aligned}$$

Ces deux suggestions ayant été obtenues à la fin de la rédaction de ce mémoire, les auteurs de [44] n'ont pas encore été contactés. Ceci sera fait prochainement afin de recueillir leur avis sur notre méthode.

Conclusion

La recherche de lois les plus creuses a beaucoup plus d'applications qu'on ne le soupçonnait au début de cette thèse. Il y a des applications "en aval" du processus de modélisation, qui entrent dans le cadre de la réduction/simplification de modèles (chapitres 2, 3 et 4). Ces applications, bien que techniques, étaient attendues. Mais le chapitre 5 montre qu'il y a aussi des applications potentielles "en amont", dans le cadre d'un logiciel d'aide à l'écriture de modèles. Au moment de l'écriture du modèle, le modélisateur pourrait s'imposer comme contrainte que l'ensemble des lois de conservation devrait avoir une base uniquement formée de lois à coefficients positifs (une sorte de critère de qualité). Les algorithmes de la thèse pourraient alors être utilisés pour chercher, parmi les modèles ayant cette propriété, ceux qui sont les plus proches de l'état courant d'un modèle en cours d'élaboration. Cette exigence de "modification minimale" rappelle celle du "principe de parcimonie", qui a des applications très fructueuses en biologie systémique, comme par exemple la "Minimization of Metabolic Adjustment" [57, section 5.5.3, Klamt, Stelling]. Les travaux de cette thèse ont donc deux prolongements très naturels : la recherche de nouvelles applications du calcul des bases les plus creuses ; et l'implantation efficace des algorithmes de la thèse dans le cadre de logiciels, non seulement de simplification de modèles, mais aussi d'aide à la modélisation.

Annexe A

Temps d'exécution des différentes versions de CSB

Modèle	m	n	$\mathcal{N}(B)$	$\mathcal{N}(B')$	$t(v_1)$	$t(v_2)$	$t(v'_1)$	$t(v'_2)$
BIOMD002	2	13	21	18	0.02	0.02	0.03	0.03
BIOMD009	7	22	44	30	4.52	2.99	0.45	0.46
BIOMD011	7	22	42	30	1.36	0.60	0.09	0.14
BIOMD013	6	27	36	33	0.46	0.25	0.15	0.16
BIOMD014	8	86	159	132	439.60	117.63	72.23	72.06
BIOMD017	6	19	32	27	2.29	0.61	2.00	0.58
BIOMD019	15	61	116	90	55.92	21.41	10.38	4.08
BIOMD023	3	13	21	17	0.04	0.04	0.03	0.04
BIOMD026	3	11	16	13	0.06	0.06	0.03	0.04
BIOMD028	3	16	22	18	0.05	0.06	0.04	0.03
BIOMD030	3	18	24	20	0.06	0.06	0.04	0.04
BIOMD038	7	17	44	28	8.60	6.44	1.65	1.70
BIOMD048	7	24	47	39	5.44	4.50	3.30	3.38
BIOMD052	3	12	20	14	0.06	0.05	0.02	0.03
BIOMD064	4	21	45	37	2.28	1.76	1.27	0.74
BIOMD066	6	11	27	20	0.71	0.75	0.17	0.19
BIOMD068	3	6	9	8	0.03	0.04	0.01	0.03
BIOMD070	8	45	52	48	1.16	0.39	0.33	0.21
BIOMD077	4	8	11	9	0.06	0.05	0.03	0.03
BIOMD085	5	17	38	32	3.64	3.67	1.39	1.33
BIOMD086	5	17	38	32	3.60	3.67	1.38	1.32
BIOMD109	4	50	52	48	0.13	0.14	0.05	0.05
BIOMD152	11	64	177	92	316.85	316.06	10.43	10.50
BIOMD153	11	75	162	111	1581.36	1580.29	75.36	75.58
BIOMD175	24	118	241	188	-	-	29786.17	13357.17
BIOMD176	6	25	38	32	0.58	0.30	0.26	0.24
BIOMD183	4	67	165	109	11.14	11.25	0.60	0.60
BIOMD186	5	11	23	18	0.70	0.59	0.14	0.16

FIGURE A.1 – Temps d'exécution des différentes versions de CSB

Modèle	m	n	$\mathcal{N}(B)$	$\mathcal{N}(B')$	$t(v_1)$	$t(v_2)$	$t(v_1')$	$t(v_2')$
BIOMD187	5	11	23	18	0.70	0.59	0.14	0.16
BIOMD193	4	8	13	12	0.12	0.14	0.12	0.14
BIOMD194	3	5	8	7	0.04	0.03	0.02	0.03
BIOMD200	7	22	39	35	0.98	0.72	0.68	0.44
BIOMD220	18	58	135	79	618.33	126.21	7.48	7.75
BIOMD223	20	84	130	97	86.54	15.73	2.24	0.94
BIOMD237	6	26	45	36	0.88	0.69	0.36	0.22
BIOMD243	3	23	17	13	0.03	0.02	0.02	0.02
BIOMD286	13	59	57	44	6.77	0.34	2.40	0.30
BIOMD305	6	10	15	14	0.28	0.17	0.29	0.18
BIOMD326	7	71	95	80	2.18	2.18	0.30	0.34
BIOMD332	16	78	253	140	55386.79	32329.70	29250.66	15226.30
BIOMD333	12	54	156	97	1014.88	979.48	564.79	371.66
BIOMD334	13	73	229	108	644.73	641.46	2.40	2.57
BIOMD336	5	18	26	22	0.18	0.10	0.05	0.05
BIOMD358	5	12	18	14	0.08	0.05	0.03	0.04
BIOMD359	3	9	14	13	0.06	0.05	0.02	0.02
BIOMD360	3	9	14	13	0.05	0.05	0.02	0.02
BIOMD361	3	8	13	11	0.04	0.04	0.02	0.02
BIOMD362	10	34	73	52	30.33	10.21	0.52	0.59
BIOMD365	4	30	49	35	0.12	0.12	0.05	0.05
BIOMD366	5	12	18	14	0.08	0.05	0.03	0.04
BIOMD384	2	6	9	8	0.02	0.02	0.01	0.01
BIOMD385	2	6	9	8	0.02	0.02	0.02	0.01
BIOMD386	2	6	9	8	0.02	0.02	0.02	0.01
BIOMD387	3	7	10	9	0.03	0.02	0.02	0.02
BIOMD409	3	17	21	20	0.05	0.04	0.03	0.04
BIOMD422	9	37	49	44	0.82	0.34	0.26	0.18
BIOMD424	14	55	73	66	3.74	0.35	2.94	0.44
BIOMD430	6	27	55	43	6.17	5.08	0.49	0.46
BIOMD431	6	27	55	43	6.15	5.07	0.48	0.45
BIOMD452	23	109	203	134	371.01	375.23	12.16	8.17
BIOMD453	23	109	203	134	371.68	375.66	12.15	8.06
BIOMD454	4	8	12	11	0.05	0.05	0.03	0.04
BIOMD455	4	9	13	12	0.04	0.05	0.03	0.05
BIOMD456	5	11	15	14	0.06	0.06	0.04	0.05
BIOMD475	7	23	48	40	24.77	14.49	25.12	14.71
BIOMD478	11	33	62	47	6.71	0.88	2.18	0.59
BIOMD488	15	69	63	50	5.16	0.33	1.79	0.41
BIOMD489	20	53	70	59	13444.08	1.61	3826.15	1.65
BIOMD494	16	80	78	70	2.98	0.36	0.21	0.42
BIOMD501	2	35	50	49	0.08	0.08	0.08	0.09
BIOMD545	6	14	13	12	0.05	0.05	0.02	0.04
BIOMD569	2	21	37	34	0.06	0.06	0.04	0.04
BIOMD574	4	195	567	317	7.81	8.32	1.41	1.43
BIOMD578	8	76	120	88	8.76	8.59	1.30	1.12

FIGURE A.2 – Temps d'exécution des différentes versions de CSB (suite)

Annexe B

Nombre de nonzéros après application d'algorithmes matriciels

Modèle	m	n	$\mathcal{N}(B)$	$\mathcal{N}(B')$	$\mathcal{N}(B_{RREF})$	$\mathcal{N}(B_{LLL})$	$\mathcal{N}(B_{HNF})$
BIOMD002	2	13	21	18	18	21	18
BIOMD009	7	22	44	30	32	30	32
BIOMD011	7	22	42	30	41	30	41
BIOMD013	6	27	36	33	35	36	50
BIOMD014	8	86	159	132	256	132	256
BIOMD017	6	19	32	27	37	32	37
BIOMD019	15	61	116	90	121	90	121
BIOMD023	3	13	21	17	18	18	18
BIOMD026	3	11	16	13	17	13	17
BIOMD028	3	16	22	18	26	18	26
BIOMD030	3	18	24	20	30	20	30
BIOMD038	7	17	44	28	36	30	36
BIOMD048	7	24	47	39	42	52	52
BIOMD052	3	12	20	14	14	15	14
BIOMD064	4	21	45	37	45	42	45
BIOMD066	6	11	27	20	22	22	20
BIOMD068	3	6	9	8	8	8	8
BIOMD070	8	45	52	48	48	48	48
BIOMD077	4	8	11	9	9	9	9
BIOMD085	5	17	38	32	36	33	36
BIOMD086	5	17	38	32	36	33	36
BIOMD109	4	50	52	48	58	48	58
BIOMD152	11	64	177	92	105	101	104
BIOMD153	11	75	162	111	147	127	135
BIOMD175	24	118	241	188	200	190	200
BIOMD176	6	25	38	32	33	38	33
BIOMD183	4	67	165	109	159	109	159
BIOMD186	5	11	23	18	20	18	20

FIGURE B.1 – Nombre de nonzéros après application de CSBprime

Modèle	m	n	$\mathcal{N}(B)$	$\mathcal{N}(B')$	$\mathcal{N}(B_{RREF})$	$\mathcal{N}(B_{LLL})$	$\mathcal{N}(B_{HNF})$
BIOMD187	5	11	23	18	20	18	20
BIOMD193	4	8	13	12	13	13	13
BIOMD194	3	5	8	7	7	7	7
BIOMD200	7	22	39	35	38	37	41
BIOMD220	18	58	135	79	89	79	89
BIOMD223	20	84	130	97	100	97	100
BIOMD237	6	26	45	36	54	38	54
BIOMD243	3	23	17	13	13	13	13
BIOMD286	13	59	57	44	61	44	61
BIOMD305	6	10	15	14	20	15	20
BIOMD326	7	71	95	80	107	80	107
BIOMD332	16	78	253	140	193	191	165
BIOMD333	12	54	156	97	132	131	114
BIOMD334	13	73	229	108	118	108	118
BIOMD336	5	18	26	22	23	22	23
BIOMD358	5	12	18	14	16	14	16
BIOMD359	3	9	14	13	16	13	16
BIOMD360	3	9	14	13	16	13	16
BIOMD361	3	8	13	11	13	11	13
BIOMD362	10	34	73	52	54	52	54
BIOMD365	4	30	49	35	43	35	43
BIOMD366	5	12	18	14	16	14	16
BIOMD384	2	6	9	8	8	9	8
BIOMD385	2	6	9	8	8	9	8
BIOMD386	2	6	9	8	8	9	8
BIOMD387	3	7	10	9	9	10	9
BIOMD409	3	17	21	20	21	20	21
BIOMD422	9	37	49	44	47	44	47
BIOMD424	14	55	73	66	71	66	71
BIOMD430	6	27	55	43	46	43	46
BIOMD431	6	27	55	43	46	43	46
BIOMD452	23	109	203	134	158	135	158
BIOMD453	23	109	203	134	158	135	158
BIOMD454	4	8	12	11	12	11	12
BIOMD455	4	9	13	12	13	12	13
BIOMD456	5	11	15	14	15	14	15
BIOMD475	7	23	48	40	68	53	66
BIOMD478	11	33	62	47	51	49	51
BIOMD488	15	69	63	50	67	50	67
BIOMD489	20	53	70	59	63	63	63
BIOMD494	16	80	78	70	70	70	70
BIOMD501	2	35	50	49	68	57	68
BIOMD545	6	14	13	12	12	12	12
BIOMD569	2	21	37	34	34	37	34
BIOMD574	4	195	567	317	482	317	482
BIOMD578	8	76	120	88	114	91	114

FIGURE B.2 – Nombre de nonzéros après application de CSBprime (suite)

Annexe C

Paramètres ayant une influence sur le temps d'exécution de la version v'_2

Modèle	m	n	d_B	$d_{B'}$	$nbEtapes$	$nbBlocs$
BIOMD002	2	13	12	9	1	1
BIOMD009	7	22	13	8	1	1
BIOMD011	7	22	10	8	1	1
BIOMD013	6	27	18	18	3	2
BIOMD014	8	86	45	38	1	1
BIOMD017	6	19	12	10	4	2
BIOMD019	15	61	13	11	5	3
BIOMD023	3	13	9	9	2	1
BIOMD026	3	11	8	5	1	1
BIOMD028	3	16	11	7	1	1
BIOMD030	3	18	13	9	1	1
BIOMD038	7	17	10	6	3	1
BIOMD048	7	24	13	12	3	1
BIOMD052	3	12	10	8	1	1
BIOMD064	4	21	18	14	2	1
BIOMD066	6	11	6	4	1	1
BIOMD068	3	6	4	3	1	1
BIOMD070	8	45	25	24	4	4
BIOMD077	4	8	4	4	2	2
BIOMD085	5	17	12	7	2	1
BIOMD086	5	17	12	7	2	1
BIOMD109	4	50	20	20	1	1
BIOMD152	11	64	32	23	6	1
BIOMD153	11	75	38	26	8	1
BIOMD175	24	118	42	32	9	5
BIOMD176	6	25	18	13	4	3
BIOMD183	4	67	61	41	1	1
BIOMD186	5	11	7	5	1	1

FIGURE C.1 – Paramètres ayant une influence sur le temps d'exécution de la version v'_2

Modèle	m	n	d_B	$d_{B'}$	$nbEtapes$	$nbBlocs$
BIOMD187	5	11	7	5	1	1
BIOMD193	4	8	4	4	2	1
BIOMD194	3	5	3	3	1	1
BIOMD200	7	22	13	11	6	5
BIOMD220	18	58	19	8	1	1
BIOMD223	20	84	27	15	12	12
BIOMD237	6	26	17	11	1	1
BIOMD243	3	23	9	6	2	2
BIOMD286	13	59	20	19	11	10
BIOMD305	6	10	4	3	2	2
BIOMD326	7	71	27	27	1	1
BIOMD332	16	78	49	26	19	2
BIOMD333	12	54	32	19	9	1
BIOMD334	13	73	50	16	2	1
BIOMD336	5	18	7	6	1	1
BIOMD358	5	12	5	4	3	3
BIOMD359	3	9	5	5	1	1
BIOMD360	3	9	5	5	1	1
BIOMD361	3	8	5	4	1	1
BIOMD362	10	34	21	9	1	1
BIOMD365	4	30	20	10	1	1
BIOMD366	5	12	5	4	3	3
BIOMD384	2	6	5	4	1	1
BIOMD385	2	6	5	4	1	1
BIOMD386	2	6	5	4	1	1
BIOMD387	3	7	5	4	2	2
BIOMD409	3	17	14	14	1	1
BIOMD422	9	37	21	17	5	5
BIOMD424	14	55	10	7	7	6
BIOMD430	6	27	15	13	1	1
BIOMD431	6	27	15	13	1	1
BIOMD452	23	109	28	13	11	5
BIOMD453	23	109	28	13	11	5
BIOMD454	4	8	4	4	1	1
BIOMD455	4	9	4	4	1	1
BIOMD456	5	11	4	4	2	2
BIOMD475	7	23	14	11	4	1
BIOMD478	11	33	11	10	6	3
BIOMD488	15	69	20	19	13	12
BIOMD489	20	53	11	11	4	4
BIOMD494	16	80	14	12	15	15
BIOMD501	2	35	27	26	2	1
BIOMD545	6	14	4	4	5	5
BIOMD569	2	21	20	17	1	1
BIOMD574	4	195	192	121	1	1
BIOMD578	8	76	28	28	5	1

FIGURE C.2 – Paramètres ayant une influence sur le temps d'exécution de la version v'_2 (suite)

Annexe D

Résultats des expérimentations avec CSBsimplexe

Modèle	m	n	$\mathcal{N}(B)$	$\mathcal{N}(B')$	$\mathcal{N}(B_{\mathcal{L}_1})$	$\mathcal{N}(B_{\mathcal{L}_2})$	t_1	t_2
BIOMD002	2	13	21	18	18	18	0.11	1.23
BIOMD009	7	22	44	30	30	30	0.70	14.4
BIOMD011	7	22	42	30	30	30	0.40	7.68
BIOMD013	6	27	36	33	35	35	0.78	26.74
BIOMD014	8	86	159	132	132	132	109.35	9489.56
BIOMD017	6	19	32	27	28	27	0.25	4.48
BIOMD019	15	61	116	90	90	90	21.83	1032.82
BIOMD023	3	13	21	17	17	17	0.10	1.16
BIOMD026	3	11	16	13	13	13	0.7	0.68
BIOMD028	3	16	22	18	18	18	0.19	2.69
BIOMD030	3	18	24	20	20	20	0.27	4.45
BIOMD038	7	17	44	28	22	28	0.15	2.58
BIOMD048	7	24	47	39	39	39	0.53	11.64
BIOMD052	3	12	20	14	14	14	0.9	0.90
BIOMD064	4	21	45	37	39	37	0.43	8.44
BIOMD066	6	11	27	20	16	20	0.5	0.54
BIOMD068	3	6	9	8	8	8	0.2	0.11
BIOMD070	8	45	52	48	48	48	4.78	214.60
BIOMD077	4	8	11	9	9	9	0.3	0.22
BIOMD085	5	17	38	32	37	32	0.19	3.11
BIOMD086	5	17	38	32	37	32	0.20	3.16
BIOMD109	4	50	52	48	48	48	17.76	592.14
BIOMD152	11	64	177	92	94	93	31.43	1529.86
BIOMD153	11	75	162	111	112	112	54.81	3921.60
BIOMD175	24	118	241	188	188	188	214.13	27651.43
BIOMD176	6	25	38	32	38	32	0.64	15.30
BIOMD183	4	67	165	109	109	109	34.90	2525.4
BIOMD186	5	11	23	18	18	18	0.6	0.59

FIGURE D.1 – Résultats des expérimentations avec CSBsimplexe

Modèle	m	n	$\mathcal{N}(B)$	$\mathcal{N}(B')$	$\mathcal{N}(B_{\mathcal{L}_1})$	$\mathcal{N}(B_{\mathcal{L}_2})$	t_1	t_2
BIOMD187	5	11	23	18	18	18	0.6	0.59
BIOMD193	4	8	13	12	12	12	0.4	0.22
BIOMD194	3	5	8	7	7	7	0.2	0.7
BIOMD200	7	22	39	35	35	35	0.42	8.16
BIOMD220	18	58	135	79	79	79	8.79	582.20
BIOMD223	20	84	130	97	97	97	42.98	4096.60
BIOMD237	6	26	45	36	36	36	1.48	35.16
BIOMD243	3	23	17	13	13	13	0.46	13.37
BIOMD286	13	59	57	44	44	44	10.66	671.50
BIOMD305	6	10	15	14	14	14	0.4	0.40
BIOMD326	7	71	95	80	80	80	34.16	2727.0
BIOMD332	16	78	253	140	140	140	59.8	4342.49
BIOMD333	12	54	156	97	97	97	13.51	573.12
BIOMD334	13	73	229	108	108	108	39.69	2654.43
BIOMD336	5	18	26	22	22	22	0.41	4.54
BIOMD358	5	12	18	14	14	14	0.7	0.76
BIOMD359	3	9	14	13	13	13	0.5	0.37
BIOMD360	3	9	14	13	13	13	0.5	0.35
BIOMD361	3	8	13	11	11	11	0.4	0.24
BIOMD362	10	34	73	52	52	52	2.37	89.9
BIOMD365	4	30	49	35	35	35	1.38	41.54
BIOMD366	5	12	18	14	14	14	0.13	1.40
BIOMD384	2	6	9	8	9	8	0.5	0.22
BIOMD385	2	6	9	8	9	8	0.2	0.13
BIOMD386	2	6	9	8	9	8	0.2	0.21
BIOMD387	3	7	10	9	10	9	0.5	0.32
BIOMD409	3	17	21	20	20	20	0.44	6.63
BIOMD422	9	37	49	44	44	44	4.21	87.98
BIOMD424	14	55	73	66	66	66	8.4	535.96
BIOMD430	6	27	55	43	43	43	1.71	23.4
BIOMD431	6	27	55	43	43	43	0.90	22.5
BIOMD452	23	109	203	134	134	134	167.13	16579.33
BIOMD453	23	109	203	134	134	134	133.23	16075.65
BIOMD454	4	8	12	11	11	11	0.6	0.40
BIOMD455	4	9	13	12	12	12	0.4	0.32
BIOMD456	5	11	15	14	14	14	0.10	1.3
BIOMD475	7	23	48	40	40	40	0.90	18.62
BIOMD478	11	33	62	47	47	47	2.68	67.23
BIOMD488	15	69	63	50	50	50	39.80	1367.57
BIOMD489	20	53	70	59	56	59	6.83	263.58
BIOMD494	16	80	78	70	70	70	56.61	3640.94
BIOMD501	2	35	50	49	49	49	5.5	86.41
BIOMD545	6	14	13	12	12	12	0.16	1.35
BIOMD569	2	21	37	34	34	34	0.47	8.38
BIOMD574	4	195	567	317	317	317	214.13	-
BIOMD578	8	76	120	88	91	88	45.49	3886.14

FIGURE D.2 – Résultats des expérimentations avec CSBsimplexe (suite)

Index

algorithme
    BasisToSolvedTask, 24
    CSBmodulo, 63
    CSBblocs, 31
    CSBprime, 31
    CSBsimplexe, 51
    CSB, 23
    EnhanceBasisUsingSolvedTask, 25
    EnhanceBasis, 23
    getSparsestFraction, 67
    getSparsestSumOfFractions, 72
    InitialisationSimplexe, 49
    IsTask, 27
    NextVector, 29
    RREFfullRanked, 45
    de Gillespie, 2
application monomiale, 60

base, 20
    de lois de conservation, 12, 38
    la plus creuse, 20
    la plus creuse modulo, 61
BioModels, 1, 7

compartiment, 2, 7
constante de réaction, 2

espèce, 2
    produit, 2
    réactant, 2

fonction d'objectif, 38

loi d'action de masse, 5
loi de conservation, 8, 10

Maple
    NullSpace, 12
matrice
    ligne-équivalente, 20
    ligne-unitaire, 26
matrice de stœchiométrie, 4
modèle, 1
    réarrangé, 7
    SBML, 7

Nicotine, 14
nonzéros, 21

objectif, 38

P-invariants, 14
polyèdre, 44
problème de démarrage, 47
programmation linéaire, 37
programme linéaire
    sous forme canonique, 47
    sous forme canonique faisable, 48
    sous forme standard, 41
propriété
    IZP, 22
    LCP, 22

réaction chimique
    irréversible, 2
réaction chimique, 2
    réversible, 2
représentant positif, 41
représentation matricielle
    d'un système d'équations, 73
    d'une fraction, 59
    d'une somme de fractions, 69

SBML, 1, 7
    libSBML, 31
simulation stochastique, 2
solution
    de base, 48
    optimale, 38
    réalisable, 38
sommet de polyèdre, 44
système
    paramétrique de fractions rationnelles, 58

tâche, 22
    initiale, 22
    résolue, 22
triplet, 59, 69

variables
    de base, 48
    hors-base, 48

Bibliographie

[1] Biomodels database. `http://www.ebi.ac.uk/biomodels-main/publmodels`, February 2016. [Online ; accessed February 8th, 2016].

[2] S. S. Andrew and D. Bray. Stochastic simulation of chemical reactions with spatial resolution and single molecule detail. *Physical Biology*, 1(3) :137, 2004.

[3] G. Auliac and J. Caby. *Mathématiques : topologie et analyse*. Objectif licence. Ediscience international, 2005.

[4] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[5] F. T. Bergmann and H. M. Sauro. SBW - a modular framework for systems biology. In *Proceedings of the 38th Conference on Winter Simulation*, WSC'06, pages 1637–1645. Winter Simulation Conference, 2006.

[6] M. Berry, M. Heath, I. Kaneko, M. Lawo, R. Plemmons, and R. Ward. An algorithm to compute a sparse basis of the null space. *Numerische Mathematik*, 47(4) :483–504, 1985.

[7] B. J. Bornstein, S. M. Keating, A. Jouraku, and M. Hucka. Libsbml : an api library for sbml. *Bioinformatics (Oxford, England)*, 24(6) :880–881, Mar. 2008.

[8] F. Boulier, M. Han, F. Lemaire, and V. G. Romanovski. Qualitative Investigation of a Gene Model Using Computer Algebra Algorithms. *Programming and Computer Software*, 41(2) :105–111, 2015. original Russian text published in Programmirovanie, 2015, vol. 41, number 2.

[9] F. Boulier, A. Korporal, F. Lemaire, W. Perruquetti, A. Poteaux, and R. Ushirobira. An Algorithm for Converting Nonlinear Differential Equations to Integral Equations with an Application to Parameter Estimation from Noisy Data. In *LNCS 8660 : Proceedings of Computer Algebra and Scientific Computing (CASC) 2014*, pages 28–43, Warsaw, Poland, 2014.

[10] F. Boulier, M. Lefranc, F. Lemaire, and P.-E. Morant. Applying a rigorous quasi-steady state approximation method for proving the absence of oscillations in models of genetic circuits. In K. H. et al., editor, *Proceedings of Algebraic Biology 2008*, number 5147 in LNCS, pages 56–64. Springer Verlag Berlin Heidelberg, 2008.

[11] F. Boulier, M. Lefranc, F. Lemaire, and P.-E. Morant. Applying a rigorous quasi-steady state approximation method for proving the absence of oscillations in models of genetic circuits. In *Prooceedings of Journées Ouvertes Biologie Informatique Mathématiques*, pages 77–82, 2008.

[12] F. Boulier, M. Lefranc, F. Lemaire, and P.-E. Morant. Model Reduction of Chemical Reaction Systems using Elimination. *Mathematics in Computer Science*, 5 :289–301, 2011.

Presented at the international conference MACIS 2007, `http://hal.archives-ouvertes.fr/hal-00184558`.

[13] F. Boulier, M. Lefranc, F. Lemaire, P.-E. Morant, and A. Ürgüplü. On proving the absence of oscillations in models of genetic circuits. In K. H. H. Anai and T. Kutsia, editors, *Proceedings of Algebraic Biology 2007*, volume 4545 of *LNCS*, pages 66–80. Springer Verlag Berlin Heidelberg, 2007. `http://hal.archives-ouvertes.fr/hal-00139667`.

[14] F. Boulier and F. Lemaire. Differential algebra and system modeling in cellular biology. In K. H. et al., editor, *Proceedings of Algebraic Biology 2008*, volume 5147 of *LNCS*, pages 22–39. Springer Verlag Berlin Heidelberg, 2008.

[15] F. Boulier, F. Lemaire, M. Petitot, and A. Sedoglavic. Chemical Reaction Systems, Computer Algebra and Systems Biology. In *Computer Algebra and Scientific Computing*, Kassel, Germany, Sept. 2011.

[16] F. Boulier, F. Lemaire, A. Ürgüplü, and A. Sedoglavic. Towards an automated reduction method for polynomial ODE models in cellular biology. *Mathematics in Computer Science, Special Issue on Symbolic Computation in Biology*, 2(3) :443–464, 2009.

[17] V. Chelliah, N. Juty, I. Ajmera, R. Ali, M. Dumousseau, M. Glont, M. Hucka, G. Jalowicki, S. Keating, V. Knight-Schrijver, A. Lloret-Villas, K. Nath Natarajan, J.-B. Pettit, N. Rodriguez, M. Schubert, S. M. Wimalaratne, Y. Zhao, H. Hermjakob, N. Le Novère, and C. Laibe. BioModels : ten-year anniversary. *Nucl. Acids Res.*, 2015.

[18] T. F. Coleman and A. Pothen. The null space problem i. complexity. *SIAM Journal on Algebraic Discrete Methods*, 7(4) :527–537, 1986.

[19] T. F. Coleman and A. Pothen. The null space problem ii. algorithms. *SIAM Journal on Algebraic Discrete Methods*, 8(4) :544–563, 1987.

[20] A. Cornish-Bowden and J.-H. S. HofMeyr. The role of stoichiometric analysis in studies of metabolism : An example. *Journal of Theoretical Biology*, 216(2) :179 – 191, 2002.

[21] G. B. Dantzig and M. N. Thapa. *Linear Programming 1 : Introduction*. Springer Series in Operations Research. Springer-Verlag New York, 1997.

[22] G. B. Dantzig and M. N. Thapa. *Linear Programming. 2. , Theory and extensions*. Springer Series in Operations Research. Springer-Verlag New York, 2003.

[23] J.-G. Dumas, C. Pernet, and Z. Sultan. Computing the rank profile matrix. In *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '15, pages 149–156, New York, NY, USA, 2015. ACM.

[24] F. Fages, S. Gay, and S. Soliman. Automatic Curation of SBML Models based on their ODE Semantics. Research Report RR-8014, INRIA, July 2012.

[25] M. Fels and P. J. Olver. Moving coframes. II. Regularization and theoretical foundations. *Acta Applicandae Mathematicae*, 55(2) :127–208, January 1999.

[26] C. Fieker and D. Stehlé. Short bases of lattices over number fields. In G. Hanrot, F. Morain, and E. Thomé, editors, *Algorithmic Number Theory*, volume 6197 of *Lecture Notes in Computer Science*, pages 157–173. Springer Berlin Heidelberg, 2010.

[27] D. T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4) :403 – 434, 1976.

[28] D. T. Gillespie. *Markov Processes : An Introduction for Physical Scientists*. Academic Press, 1992.

[29] D. T. Gillespie. A rigorous derivation of the chemical master equation. *Physica A : Statistical Mechanics and its Applications*, 188(1–3) :404 – 425, 1992.

[30] R. Heinrich and S. Schuster. *The Regulation Of Cellular Systems*. Springer, Aug. 1996.

[31] V. Henri. *Lois générales de l'Action des Diastases*. Hermann, Paris, 1903.

[32] S. Hoops, S. Sahle, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. COPASI—a COmplex PAthway SImulator. *Bioinformatics*, 22(24) :3067–3074, Dec. 2006.

[33] E. Hubert and G. Labahn. Scaling invariants and symmetry reduction of dynamical systems. *Foundations of Computational Mathematics*, 13(4) :479–516, 2013.

[34] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J.-H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. L. Novère, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, J. Wang, and S. B. M. L. Forum. The systems biology markup language (sbml) : a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4) :524–531, Mar. 2003.

[35] N. Juty, R. Ali, M. Glont, S. Keating, N. Rodriguez, M. J. Swat, S. M. Wimalaratne, H. Hermjakob, N. Le Novère, C. Laibe, and V. Chelliah. BioModels : Content, Features, Functionality and Use. *CPT : Pharmacometrics and Systems Pharmacology*, 2015.

[36] V. Klee and G. J. Minty. How good is the simplex algorithm ? In O. Shisha, editor, *Inequalities*, volume III, pages 159–175. Academic Press, New York, 1972.

[37] N. Le Novère, B. Bornstein, A. Broicher, M. Courtot, M. Donizelli, H. Dharuri, L. Li, H. Sauro, M. Schilstra, B. Shapiro, J. L. Snoep, and M. Hucka. BioModels Database : a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Research*, 34(Database issue) :D689–D691, Jan 2006.

[38] F. Lemaire and A. Temperville. On defining and computing "good" conservation laws. In P. Mendes, J. Dada, and K. Smallbone, editors, *Computational Methods in Systems Biology*, volume 8859 of *Lecture Notes in Computer Science*, pages 1–19. Springer International Publishing, 2014.

[39] F. Lemaire and A. Ürgüplü. Mabsys : Modeling and analysis of biological systems. In *Proceedings of ANB2010*, pages 57–72, 2010.

[40] F. Lemaire and A. Ürgüplü. A method for semi-rectifying algebraic and differential systems using scaling type Lie point symmetries with linear algebra. In A. Press, editor, *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, 2010.

[41] A. K. Lenstra, H. W. Lenstra, and L. Lovasz. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4) :515–534, 1982.

[42] C. Li, M. Donizelli, N. Rodriguez, H. Dharuri, L. Endler, V. Chelliah, L. Li, E. He, A. Henry, M. I. Stefan, J. L. Snoep, M. Hucka, N. Le Novère, and C. Laibe. BioModels Database : An enhanced, curated and annotated resource for published quantitative kinetic models. *BMC Systems Biology*, 4 :92, June 2010.

[43] C. E. Martin, C.-S. S. Oh, P. Kandasamy, R. Chellapa, and M. Vemula. Systems biology markup language : Level 2 and beyond. *Biochemical Society transactions*, 31(Pt 6) :1472–1473, Dec. 2003.

[44] S. I. Martins and M. A. V. Boekel. Kinetic modelling of Amadori N-(1-deoxy-D-fructos-1-yl)-glycine degradation pathways. Part II—kinetic analysis. *Carbohydrate Research*, 338(16) :1665 – 1678, 2003.

[45] P. Mendes, S. Hoops, S. Sahle, R. Gauges, J. Dada, and U. Kummer. *Systems Biology*, chapter Computational Modeling of Biochemical Networks Using COPASI, pages 17–59. Humana Press, Totowa, NJ, 2009.

[46] L. Michaëlis and M. Menten. Die kinetik der invertinwirkung (the kinetics of invertase activity). *Biochemische Zeitschrift*, 49 :333–369, 1973. Partial english translation available on `http://web.lemoyne.edu/~giunta/menten.html`.

[47] P. J. Olver. *Applications of Lie groups to differential equations*, volume 107 of *Graduate Texts in Mathematics*. Springer Verlag, second edition, 1993.

[48] M. A. Panteleev, M. V. Ovanesov, D. A. Kireev, A. M. Shibeko, E. I. Sinauridze, N. M. Ananyeva, A. A. Butylin, E. L. Saenko, and F. I. Ataullakhanov. Spatial propagation and localization of blood coagulation are regulated by intrinsic and protein C pathways, respectively. *Biophysical Journal*, 90(5) :1489 – 1500, 2006.

[49] Péter Érdi and János Tóth. *Mathematical models of chemical reactions : theory and applications of deterministic and stochastic models*. Princeton University Press, 1989.

[50] S. Roman. *Advanced Linear Algebra*. Graduate Texts in Mathematics. Springer, 2007.

[51] H. M. Sauro and B. Ingalls. Conservation analysis in biochemical networks : computational issues for software writers. *Biophysical Chemistry*, 109(1) :1–15, 2004.

[52] A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.

[53] A. Sedoglavic. Reduction of algebraic parametric systems by rectification of their affine expanded Lie symmetries. In H. Anai, K. Horimoto, and T. Kutsia, editors, *Proceedings of Algebraic Biology* 2007 – *Second International Conference*, volume 4545 of *Lecture Notes in Computer Science*, pages 277–291, RISC, Castle of Hagenberg, Austria, July 2–4 2007. Springer.

[54] S. Soliman. Nicotine software. `https://contraintes.inria.fr/~soliman/nicotine.html`. [Online ; accessed February 8th, 2016].

[55] S. Soliman. Invariants and Other Structural Properties of Biochemical Models as a Constraint Satisfaction Problem. *Algorithms for Molecular Biology*, 7(1) :15, 2012.

[56] T. Sturm and A. Weber. *Algebraic Biology : Third International Conference, AB 2008, Castle of Hagenberg, Austria, July 31-August 2, 2008 Proceedings*, chapter Investigating Generic Methods to Solve Hopf Bifurcation Problems in Algebraic Biology, pages 200–215. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[57] Z. Szallasi, J. Stelling, and V. Periwal. *System Modeling in Cellular Biology : From Concepts to Nuts and Bolts*. The MIT Press, 2006.

[58] B. Teusink, M. C. Walsh, K. van Dam, and H. V. Westerhoff. The danger of metabolic pathways with turbo design. *Trends in Biochemical Sciences*, 23(5) :162 – 169, 1998.

[59] R. R. Vallabhajosyula, V. Chickarmane, and H. M. Sauro. Conservation analysis of large biochemical networks. *Bioinformatics*, 22(3) :346–353, 2006.

[60] S. Vidal, M. Petitot, F. Boulier, F. Lemaire, and C. Kuttler. Models of stochastic gene expression and Weyl algebra. In *Proceedings of ANB2010*, pages 76–97, 2010.

Résumé

Les nouveaux algorithmes présentés dans cette thèse contribuent à la thématique générale de la simplification des modèles biologiques : le calcul de bases creuses de lois de conservation, la simplification des systèmes d'équations différentielles paramétriques, fréquents en modélisation, et le reverse engineering des modèles.

Les algorithmes de cette thèse sont basés sur de l'algèbre linéaire exacte. Le chapitre 2 présente un algorithme glouton exact et garanti permettant de calculer une base la plus creuse parmi toutes les bases d'un espace vectoriel. On l'applique au calcul de lois de conservation de modèles biologiques. Dans le chapitre 3, une variante de cet algorithme utilise la résolution de plusieurs programmes linéaires (avec l'algorithme du simplexe) en variables réelles. Cette variante permet de calculer des bases creuses sans garantir qu'elles soient complètes ou les plus creuses. Le chapitre 4 présente un algorithme permettant de calculer une base la plus creuse modulo un espace vectoriel. Il permet de simplifier des fractions rationnelles en effectuant des changements de variables. Enfin, le chapitre 5 présente un algorithme qui, dans le cas où l'ensemble des lois de conservation d'un modèle biologique n'admet pas de base complète de lois à coefficients positifs, suggère des modèles enrichis d'une ou plusieurs espèces.

Mots-clés: calcul formel, modélisation en biologie, loi de conservation, base creuse d'un espace vectoriel.

Abstract

The new algorithms introduced in this thesis contribute to the general theme of simplification of biological model : the computation of sparse bases of conservation laws, the simplification of parametric systems of differential equations, frequent in modelling, and the reverse engineering of models.

The algorithms of this thesis are based on exact linear algebra. The chapter 2 introduces a greedy, exact and guaranteed algorithm which allows to compute a sparsest basis among all the bases of a vector space. We apply it to the computation of conservation laws of biological models. In the chapter 3, a variant of this algorithm uses the resolution of several linear programs (with the simplex algorithm) in real variables. This variant allows to compute sparse bases without guarantee they are complete or sparsest. The chapter 4 introduces an algorithm which allows to compute a sparsest basis modulo a vector space. It was developed with the aim to simplify rational fractions using changes of variables. The chapter 5 introduces an algorithm which suggests models enriched of one or several species, in the case where the set of conservation laws does not admit a complete basis of laws with nonnegative coefficients.

Keywords: symbolic computation, biological modelling, conservation law, sparse basis of a vector space.