Mémoire : Construction d'algorithmes pour résoudre des systèmes d'équations différentielles non linéaires.

Frederic Pluquet

4 mai 2004

Remerciements

J'aimerai remercier ici les personnes qui ont contribué, de près ou de loin, que ce soit par une aide morale ou à la compréhension de la matière traitée ou par un appui énergique, à l'aboutissement de ce mémoire.

Tout d'abord, *Madame Jacqueline Sengier*, grâce à qui j'ai franchi le seuil de cette université et qui m'a fait confiance en me proposant ce sujet de mémoire. Je voudrais tout particulièrement la remercier pour sa dévotion aux autres sans attendre quelque chose en retour, pour son immense générosité et sa gentillesse sans limite.

Monsieur Pierre Bieliavsky, qui a fait de son métier sa passion, pour ses idées dont lui seul a le secret.

Monsieur Léon Brenig, pour sa rigueur mathématique et son envie de toujours avancer ainsi que pour sa gentillesse et ses nombreux encouragements.

Monsieur Marco Codutti, pour ses encouragements et son oeil avisé d'informaticien.

Monsieur Alain Gottcheiner, pour son oeil correctif.

Mademoiselle Magali Molle, qui a supporté bon nombre de mes états d'âme et qui a su me sortir la tête du sable quand il le fallait.

Monsieur et Madame Mes Parents, qui m'ont permis de faire ces deux années d'étude supplémentaires et qui m'ont offert un cocon idéal pour travailler en toute sérénité.

Si j'oublie quelqu'un dans cette page, qu'il ne m'en veuille pas et qu'il en soit remercier au centuple.

Table des matières

Chapitre 1

Introduction

"Un mémoire sur des équations mathématiques ? Moi ? Je ne connais pas grand chose dans cette matière et je ne connais absolument pas le sujet de ce mémoire." furent mes premières pensées à l'écoute du sujet que me proposait Madame Sengier en novembre dernier. J'avais vraiment des doutes quant à un aboutissement heureux de ce travail.

Mais la confiance et la patience des hommes et femmes qui m'ont entouré tout au long de ce projet m'ont permis de découvrir un monde que je ne connaissais pas, la recherche mathématique. Ils ont réussi à me faire connaître, voire aimer, ce qui les passionnait. C'est tellement agréable de travailler avec des personnes qui ont la même passion devant une équation différentielle qu'un enfant devant un nouveau jouet. Cela me fascinait à un tel point qu'on puisse autant aimer cela, que j'ai essayé de les comprendre et d'entrer dans leur monde. Ils ont été les cerveaux et moi, leurs mains de traducteur sur le clavier.

1.1 La recherche

Il n'est pas simple d'entrer dans le monde de la recherche. Les hommes et femmes qui y vivent sont pris continuellement d'une envie d'avancer comme si un flux les poussait dans le dos.

Au premier abord, vous avez l'impression que le vent vous est contraire. Il est très, très difficile de comprendre le langage des chercheurs. Mais un jour, vous vous étonnez que tous les mots employés cachent de définitions simples et que ces mots forment une certaine base d'échange entre vous et les chercheurs. Un langage commun est alors établi et le vent tourne. Vous comprenez alors que la lueur dans leurs yeux reflète l'ambition d'étudier le monde environnant, en particulier, ici, via les mathématiques.

Bien des personnes m'ont demandé quel était le sujet de ce mémoire. Une fois celui-ci énoncé, la perplexité s'intallait sur leurs visages. J'expliquais alors que ces systèmes d'équations permettaient simplement de décrire mathématiquement tout ce qui nous entoure : les variations de température, la pression, la gravitation, ... Les domaines qu'ils recouvrent sont très vastes : de la physique aux sciences politiques, en passant par la chimie, la biologie, l'astronomie, ... Tout, ou presque, peut être représenté par ces équations.

Le problème de ces équations est qu'on ne peut les résoudre, c'est-à-dire trouver une groupe de valeurs qui permettent de rendre vraie chaque équation du système présenté, qu'avec des méthodes non linéaires. Nous verrons plus en détail quelles complications amènent la non-linéarité par rapport à la linéairité.

C'est à cet instant que l'informatique et les chercheurs en mathématiques se rencontrent. Ces derniers ont beaucoup d'idées sur la manière de résoudre partiellement ces systèmes. Mais la lourdeur des calculs empêche parfois d'achever un raisonnement. L'informatique permet de résoudre ce problème.

Afin d'avancer au mieux dans ce mémoire, quatre chercheurs et professeurs et moi nous sommes réunis pour échanger des idées environ une fois par semaine.

Ces quatre personnes sont :

- Madame *Jacqueline Sengier*, chercheur et professeur en mathématique et informatique,
- Monsieur *Pierre Bieliavsky*, chercheur et professeur en mathématique,
- Monsieur *Léon Brenig*, chercheur et professeur en physique,
- Monsieur *Marco Codutti*, professeur en informatique.

Pour réussir à étudier et décortiquer une large classe de systèmes d'équations différentielles, nous nous sommes basés sur de nouvelles idées de ces derniers. Nous verrons au cours de ce mémoire que d'autres études ont été réalisées. Nous expliquerons partiellement la méthode théorique et la méthode basée sur Riemann. Ce domaine étant encore très peu connu, il est difficile de faire des liens directs entre chacune de ses méthodes. Nous verrons que ces différentes recherches aboutissent à des résultats n'ayant pas exactement les mêmes contraintes d'application. Elles pourraient peut-être se révéler complémentaires.

1.2 Maple, NODES et moi

Maple est un logiciel permettant de travailler sur des objets mathématiques (matrices, équations, vecteurs,...) et d'y appliquer des fonctions, le tout sous forme aussi bien numérique que formelle. *Maple* est l'outil idéal pour effectuer des calculs lourds et fastidieux.

NODES est une librairie, originellement écrite pour *MATHEMATICA* et totalement réécrite pour *Maple*, développée par Marco Codutti en 1990. Cette librairie permet de travailler sur une large classe de systèmes d'équations non linéaires. On peut, par exemple, transformer très simplement un système d'équations non linéaires en une équation du type Lokta-Volterra.

Ce mémoire est une sorte de suite à *NODES*. Nous allons essayer, tout au long de ce mémoire, de découvrir les équations différentielles, les différentes manières de les représenter et nous allons surtout nous attarder sur le moyen de simplifier celles-ci afin de trouver plus facilement des solutions pour un système d'équations, et ce de manière automatisée.

Nous allons étudier deux cas : le cas de dimension 2 et de dimension 3.

1.3 Contenu du mémoire

Nous commencerons par expliquer ce que sont véritablement *Maple*, *NODES* et en quoi ce mémoire contribue à améliorer le programme *NODES* tel qu'il se présente aujourd'hui.

Nous verrons une brève introduction sur les équations différentielles.

Ensuite, nous retracerons l'acheminement des idées apparues tout au long de cette recherche et nous verrons la théorie s'y rattachant.

Les algorithmes accompagnant cette théorie seront expliqués dans un autre chapitre que celle-ci afin de ne pas la surcharger.

De plus, l'ensemble du code produit et les résultats volumineux sont joints en annexe.

1.4 Notations

Plusieurs notions seront utilisées dans ce mémoire afin, le plus souvent, d'éclaicir les équations. Celles-ci sont décrites ci-dessus.

1.4.1 Les notations générales

Toute référence bibliographique est représentée par un numéro entre [ ] renvoyant à la page 68.

1.4.2 Les notations mathématiques

- $\dot{x}$ représente la dérivée de x par rapport à la variable indépendante ;
- Si elle n'est pas clairement redéfinie avant son utilisation, $\delta_{i,j}$ représente la fonction définie comme suit :

$$\begin{cases} \delta_{i,j} = 1, \, si \, i = j \\ \delta_{i,j} = 0, \, si \, i \neq j \end{cases}$$

1.4.3 Les notations de programmation

Dans les descriptions générales de concepts de programmation, nous pouvons retrouver certaines de ces notations :

- Un texte entre crochets (**[***texte***]**) est optionnel dans la commande ;
- **{** *choix1* | *choix2* | *choix3* **}** représente une suite de choix exclusifs.

Chapitre 2

Maple, NODES et programmation

Toute l'implémentation des algorithmes présentés dans ce mémoire a été testée sous *Maple* 9[1], sans *NODES* pour cause d'incompatibilité avec *Maple* (en effet, comme nous le verrons, *NODES* a été réécrit pour *Maple* 5 et non la version 9). Pour des explications plus approfondies sur *NODES*, se reporter au manuel utilisateur l'accompagnant ou au mémoire de Marco Codutti[10].

Nous commencerons donc par une introduction à *Maple*, ses capacités et son utilisation. Cette introduction est basée sur l'aide inclue dans *Maple* et sur les livres d'André Heck et M.B. Monagan[8, 9]. Nous continuerons sur un brève introduction de *NODES*.

2.1 Introduction à *Maple*

Développé par des chercheurs de l'université ETH de Zurith et de celle de Waterloo, au Canada, *Maple* est un logiciel qui permet de manipuler des objets mathématiques, en travaillant sous forme symbolique, numérique ou graphique.

Maple est assez répandu car il est simple à mettre en action et très puissant. Puissant par son moteur algébrique et simple car son interface utilisateur est assez intuitive et conviviale. Son aide est très bien réalisée, accompagnée d'un moteur de recherche permettant de trouver quasi directement ce que l'on cherche.

Maple permet d'exporter facilement des sessions de travail (équations, résultats, ...) vers différents formats tels que LaTeX ou HTML avec ou sans *MathML*. Mais il est à noter que le LaTeX exporté par *Maple* ne peut être incorporé directement dans un autre docu-

[1] *Maple 9.01, Single User Profile, copyright(c) 1981-2003 by Maplesolf.*

ment LaTeX via Lyx (un éditeur graphique pour LaTeX utilisé pour la conception de ce mémoire). En effet, des erreurs apparaissent très vite. De plus, ce code n'est pas facilement compréhensible et utilise des librairies propres à *Maple*. Bref, cette fonction a très peu servi à la rédaction de ce mémoire.

Maple possède surtout son propre langage de programmation. Nous décrivons ici l'utilisation de base de celui-ci afin de mettre au point les différentes notions utilisées plus tard dans le code.

2.1.1 L'environnement de travail

Maple fournit également un très bon environnement de travail pour manipuler le langage et ses résultats.

L'environnement se présente comme un *shell* qui prend en entrée une expression complète, calcule celle-ci et affiche le résultat, s'il y en a un. Ensuite, il attend de nouveau une expression et ainsi de suite.

Pour référencer le dernier résultat calculé, on peut utiliser le symbole %. De la même manière, %% et %%% représentent l'avant-dernier et l'antépénultième résultats.

Maple est "case sensitive", c'est-à-dire que *Diff* et *diff* sont 2 fonctions différentes à son regard.

La commande *?topic* permet d'appeler l'aide sur une commande particulière. Nous pouvons prendre comme exemple *?type* qui nous affichera l'aide sur la commande *type*.

La commande *restart* permet de remettre à zéro la mémoire des variables et procédures utilisateur de *Maple*.

Si une commande se termine par le symbole " ; ", le résultat sera affiché sur la ligne qui suit.

Par contre, si elle se termine par le symbole " : ", le résultat sera calculé mais pas affiché.

Un texte compris entre les caractères "##" et la fin de la ligne est considéré comme un commentaire.

2.1.2 Les variables

Déclaration et assignation

Pour déclarer une variable, il suffit de lui donner un nom et une valeur initiale.

Exemple :

```
> a := 100/47 ;
```

$$a := \frac{100}{47}$$

Libération

Si vous voulez libérer le nom d'une variable, il faut utiliser la commande *unassign*.

Exemple :

```
> a := 100/47 ;
```

$$a := \frac{100}{47}$$

```
> unassign('a') ;
> a ;
```

$$a$$

La commande *restart* permet de les libérer toutes (mais en même temps que les procédures enregistrées).

Constantes et variables prédéfinies

Certains objets sont prédéfinis au démarrage de *Maple*. Toutes les constantes prédéfinies peuvent être affichées via la commande *constantes*.

Exemple :

```
> constantes ;
```

$$false, \gamma, \infty, \ true, \ Catalan, \ FAIL, \ \pi$$

2.1.3 Les types

Types de base

Le tableau 2.1 reprend, de façon non exhaustive, les types de base utilisés dans *Maple*.

TAB. 2.1 – Types

Type	Description
numeric	N'importe quel nombre
integer	Un nombre entier positif ou négatif
posint	Un nombre entier positif
negint	Un nombre entier négatif
complex	Un nombre complexe
float	Un nombre réel
fraction	Une fraction
string	Une chaîne de caractère
procedure	Une procédure
boolean	Un booléen qui prend les valeurs false ou true.
set	Un ensemble de valeurs entre {}
symbol	Un symbole utilisé(variable, procédure,...)

Comparaison

La commande *type* permet de comparer un symbole avec un type donné. Celle-ci renvoie une valeur booléenne.

Exemple :

```
> type(100/47, fraction) ;
```

true

```
> type(100/47, integer) ;
```

false

La commande *whattype* nous renseigne sur le type d'un objet.

Exemple :

```
> whattype(100/47) ;
```

fraction

Les tableaux

Les tableaux sont des structures à n dimensions d'hyper-rectangles. Ceux-ci sont construits sous *Maple* grâce au type *Array*.

Il faut savoir que la numérotation des indices d'un tableau commence par 1 et non par 0, comme dans beaucoup de langages de programmation usuels.

Chaque élément d'un tableau peut accepter comme valeur n'importe quel objet. L'opérateur " :=" permet d'assigner une valeur à

un élément d'un tableau.

Exemple :

```
## Tableau à 4 dimensions
> a := Array(1..2,1..2,1..2,1..5) :
> a[1,1] ;
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

```
> a[1,1,2,3] := 5 ;
```

$$a_{1,1,2,3} := 5$$

```
> a[1,1] ;
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \end{bmatrix}$$

Définition de nouveaux types

Maple permet de définir de nouveaux types d'objets. Un nouveau type peut être déclaré de deux manières différentes :

1. `‘type/NomType‘ := SetOfTypes`

   Exemple :

   ```
   ## reconnaître les nombres et les booléens
   > ‘type/NUMBOOL‘ := {numeric, boolean} ;
   ```

2. `‘type/NomType‘ := proc(objet_a_comparer [,arg1[,...[,argN]]])`
   `[BLOC]`
   `end ;`

   Exemple :

   ```
   > ‘type/LTy‘ := proc(x,y)
   >                 if(x < y) then
   >                    true ;
   >                 else
   >                    false ;
   >                 end if ;
   >                 end :
   > type(4,LTy(5)) ;
   ```

   true

2.1.4 Les structures de contrôles

Comme dans beaucoup de langages de programmation, des opérateurs sont présents pour contrôler la séquence d'éxécution des instructions. Il n'est pas nécessaire de s'attarder en détail sur l'explication de ceux-ci. La syntaxe et un exemple pour ces deux opérateurs suffiront aisément à la compréhension de ces derniers.

if

Syntaxe

```
if condition then
     [expressions_1]
 [elif condition then
     [expressions_2]]
 [else
     [expressions_3]]
 end if
```

Exemple

```
> a := 3 ; b := 5 ;
```

$$a := 3$$

$$b := 5$$

```
> if (a > b) then a else b end if ;
```

$$5$$

for

Syntaxe

for *name* from *expression_1* [by expression_2] to *expression_3* do
     [expressions]
 end do ;

Exemple

```
> for i from 6 by 2 to 100 do print(i) end do ;
 ## imprime tous les chiffres paires compris entre 6 et 100
```

2.1.5 Les procédures

Noyau de base

Dès le démarrage de Maple, un ensemble de procédures est directement accessible. Elles sont très nombreuses. Je ne donnerai donc ici que quelques exemples choisis.

Exemples :

```
> min(400,51,89) ;
```

51

```
> log(3) ;
```

$$\ln(3)$$

```
> evalf(35 + ln(3)) ;
```

$$36.09861229$$

```
> exp(4) ;
```

$$e^4$$

L'aide de *Maple* comporte une recherche par mots clés qui permet de retrouver facilement des fonctions courantes.

Fonctions particulièrement utiles pour ce mémoire

- *add*
  La fonction *add* permet comme son nom l'indique de sommer sur une séquence de valeurs. Cette fonction sera énormément utilisée dans ce mémoire. En effet, beaucoup de formules décrites ici comprendront des sommations.

  Exemple

  ```
  > add(i*i,i=1..5)
  ```

  55

- *subs*
  Cette fonction permet de substituer une expression par une autre dans une troisième expression. Cette fonction est particulièrement utile lorsque vous voulez donner une valeur à une variable dans une équation et en calculer directement le résultat.

  Exemple

  ```
  > subs(x=5,y=6, x^2+y*3-5=z) ;
  ```

$$38{=}z$$

– *simplify*
Voici une fonction qui permet de simplifier au maximum une expression donnée. Ces simplifications sont basées sur des règles de toutes sortes.

Exemple
```
> simplify(exp(a+ln(b*exp(c)))) ;
```

$$b\,e^{(a+c)}$$

Définition de nouvelles procédures

Nous pouvons définir de nouvelles procédures par la syntaxe suivante :

```
NomProcedure := proc( [arg1 [ : : type]
                      [, arg2 [ : : type]
                      [, ...
                      [, argn [ : : type]]]])
      [expressions]
       end ;
```

Le résultat de la dernière expression calculée dans le corps de la procédure est renvoyé à la fin de l'exécution.

L'argument "type" d'un argument de la procédure n'est pas obligatoire mais est utile lorsque l'on veut être sûr du type employé.

Exemple :
```
> Square := proc ( x : : integer )
             x * x ;
            end ;
```

Square := **proc** (x) x*x **end proc**

```
> Square(5) ;
```

25

```
> Square(5.2) ;
```

Error, invalid input : Square expects its 1st argument, x, to be of type integer, but received 5.2

2.1.6 Les librairies

Utilité

Les librairies permettent d'assembler plusieurs déclarations de variables, types ou procédures dans un même fichier. C'est le cas de *NODES* et du code résultant de ce mémoire.

Librairies standard

Maple offre, dès l'installation, une série de librairies dites "standard". Elles sont, elles aussi, très nombreuses. Je ne donnerai, ici encore, que quelques exemples choisis.

La libraire *linalg* contient toutes les fonctions utilisées en l'algèbre linéaire et des calculs de vecteurs. Dans celle-ci, nous retrouvons *matrix* ou *Matrix*, qui permettent de créer des matrices avec plus ou moins d'options, *coldim*, qui nous renseigne sur le nombre de colonnes d'une matrice, ou encore *jacobian*, qui calcule la matrice jacobienne d'un vecteur de fonctions.

La commande *with* permet de préciser avec quelles librairies nous voulons travailler.

Exemple :

```
> with(linalg) ;
```

Librairies additionnelles

Il est aisé de définir de nouvelles librairies, à l'instar de *NODES*. Il vous suffit de mettre toutes les fonctions que vous désirez dans un fichier texte. Ces librairies peuvent être chargées grâce à la commande *read* suivie du chemin d'accès de la librairie choisie.

Exemple :

```
> read "/usr/local/nodes/nodes" ;
```

2.2 Introduction à *NODES*

Non linear Ordinary Differential Equations Solver (*NODES*) a vu le jour un beau matin de l'année 1990 avec la conception du mémoire de Marco Codutti[10]. L'idée était d'informatiser des concepts et des idées pour résoudre des équations différentielles ordinaires non linéaires (ci-après EDO). Ces idées sont, quant à elles, vieilles seulement d'une année de plus. Elles ont été décrites et formalisées par Alain Goriely et Léon Brenig, dans un mémoire de l'ULB[11].

NODES devint vite un outil très précieux à la recherche des solutions d'EDO. Ecrit à la base pour *MATHEMATICA* (autre logiciel de calcul symbolique, concurrent de *Maple*), *NODES* est réécrit par Marco Codutti pour *Maple V release 3*.

NODES est donc une librairie pour *Maple*. Une fois chargée, elle offre une foule de fonctions. Elle supporte, entre autres, 2 représentations de systèmes d'EDO.

Au cours du développement de ce mémoire, Marco Codutti et moi avons découvert que *NODES* ne fonctionnait pas du tout sous *Maple 9*. En effet, beaucoup de fonctions ne sont pas référencées de la même manière dans les deux versions de *Maple*. Nous avons passé beaucoup de temps à essayer d'en comprendre la raison et avons estimé que le travail de réécriture serait long et assez important. La suite du mémoire a été écrite comme si *NODES* fonctionnait. En effet, les deux librairies sont complémentaires et indépendantes l'une de l'autre.

2.3 Comment charger le programme de ce mémoire sous *Maple* ?

Le programme résultant de ce mémoire est une librairie. Il suffit donc de la copier quelque part sur votre disque dur puis d'exécuter dans *Maple* l'instruction suivante :

```
> read "<CHEMIN_DE_LA_LIBRAIRIE>/programmation memoire";
```

dans laquelle `<CHEMIN_DE_LA_LIBRAIRIE>` doit être remplacé par le chemin d'accès à la librairie à partir de la racine de votre disque dur.

Chapitre 3

Introduction aux équations différentielles

Nous présentons ici une introduction aux équations différentielles, afin de fixer certaines notations et de présenter certains concepts qui seront utilisés dans la suite de ce mémoire. Nous nous baserons essentiellement sur des exemples. Nous finirons par établir le lien avec le type de systèmes d'équations qui nous intéressera particulièrement par la suite.

3.1 Définitions

Nous commençons par quelques définitions de base des termes employés dans la suite de cet ouvrage.

- Equation différentielle

  Une équation différentielle est une équation qui décrit les relations entre une ou plusieurs fonctions inconnues et leurs dérivées.

- Equation différentielle ordinaire (EDO)

  Soient $y = y(x)$ et ses dérivées $\frac{dy}{dx}, \frac{d^2y}{dx^2}$, ..., $\frac{d^ny}{dx^n}$.
  Par définition, une équation différentielle ordinaire est une équation liant x, y, y', y'', ...,$y'^{(n)}$
  Par exemple : $y'^{(n)} = F(x, y(x), y'(x), .., y'^{(n-1)})$

- Equations différentielles linéaire et non linéaire

  Si $y'^{(n)} = F(x, y(x), y'(x), .., y'^{(n-1)})$ est une équation linéaire,

alors la solution générale $y(x)$ dépend de n constantes arbitraires. Toutes les solutions d'une équation différentielles linéaires sont obtenues en imposant des conditions sur ces constantes.

Si $y'^{(n)} = F(x, y(x), y'(x), .., y'^{(n-1)})$ est une équation non linéaire, alors la solution générale contient également n constantes d'intégration mais il peut exister des solutions supplémentaires de ces équations non linéaires, que l'on n'obtient pas par un choix de constantes d'intégration.

- ORDRE

  L'ordre d'une équation différentielle est donné par l'ordre le plus élevé des dérivées la composant.

  Exemple : $y'' = \frac{yy'}{x}$ est d'ordre 2.

- SYSTÈME D'ÉQUATIONS

  Un système d'équations est un ensemble d'équations couplées.

3.2 Au coeur des équations différentielles

3.2.1 Exemples généraux

Voici quelques exemples d'équations différentielles rencontrées souvent en physique, chimie et dans d'autres domaines.

1. Prenons comme premier exemple, la loi de Newton : $F = m.a$ où $a = \frac{d^2x}{dt^2}$.
   Nous pouvons ramener cette équation à une équation différentielle du second ordre :
   $$\frac{d^2x}{dt^2} = \frac{F(x)}{m}$$

2. La loi de décroissance radioactive est une équation différentielle ordinaire du premier ordre.
   Le pourcentage de décroissance est une constante dépendante du matériau initial (uranium, plutonium, ...) ; il est donné par la formule :

$$\frac{\frac{dx}{dt}}{x} = -k \Rightarrow \frac{dx}{dt} = -k.x$$

La solution est $x(t) = x(0).e^{-kt}$.

3.2.2 Systèmes d'équations différentielles

Un grand nombre de systèmes d'équations différentielles peut être mis sous la forme de systèmes d'équations différentielles du premier ordre, c'est-à-dire :

$$\begin{cases} \frac{dx_1}{dt} = f_1(t, x_1, x_2, ..., x_n) \\ \frac{dx_2}{dt} = f_2(t, x_1, x_2, ..., x_n) \\ \qquad\qquad \vdots \\ \frac{dx_n}{dt} = f_n(t, x_1, x_2, ..., x_n) \end{cases}$$

dans lequel $f_1, f_2, ..., f_n$ sont des fonctions suffisamment régulières. Si celles-ci sont toutes linéaires, le système sera également dit *linéaire*.

Pour transformer une équation différentielle de la forme

$$a_n(t)\frac{d^n y}{dt^n} + a_{n-1}(t)\frac{d^{n-1} y}{dt^{n-1}} + ... + a_0 y = 0$$

en un système de n équations du premier ordre, on effectue les opérations suivantes.

Soit $x_1(t) = y$, $x_2(t) = \frac{dy}{dt}$,... et $x_n(t) = \frac{d^{n-1}y}{dt^{n-1}}$. On obtient alors :

$$\begin{cases} \frac{dx_1}{dt} = x_2, \frac{dx_2}{dt} = x_3, ..., \frac{dx_{n-1}}{dt} = x_n \\ \frac{dx_n}{dt} = -\frac{a_{n-1}(t)x_n + a_{n-2}(t)x_{n-1} + ... + a_0 x_1}{a_n(t)} \end{cases}$$

Le système le plus général de n équations linéaires du premier degré a la forme suivante :

$$\begin{aligned} \frac{dx_1}{dt} &= a_{11}(t)x_1 + a_{12}(t)x_2 + ... + a_{1n}(t)x_n + g_1(t) \\ &\vdots \\ \frac{dx_n}{dt} &= a_{n1}(t)x_1 + a_{n2}(t)x_2 + ... + a_{nn}(t)x_n + g_n(t) \end{aligned} \tag{3.1}$$

Soit le vecteur $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ et la matrice $A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$.

Le système 3.1 peut alors être vu sous la forme :

$$\dot{x} = \frac{dx}{dt} = Ax$$

Exemple : Modèle de l'oscillateur

Le modèle de l'oscillateur harmonique, une masse soumise à la force exercée par un ressort, est gouverné par l'équation :

$$\frac{d^2x}{dt^2} = -\frac{k}{M}x(t)$$

où M représente la masse de l'objet et k la constante de régidité du ressort.

On pose $\frac{dx}{dt} = v$.
Donc

$$\frac{d}{dt}(\frac{dx}{dt}) = -\frac{k}{M}\,x$$

ou encore :

$$\left\{ \begin{array}{rcl} \frac{d}{dt}x & = & v \\ \frac{d}{dt}v & = & -\frac{k}{M}x \end{array} \right.$$

Nous avons donc bien ramené une équation du second ordre à un système de deux équations du premier ordre.
Passage à l'écriture matricielle

$$\frac{d}{dt}\left[\begin{array}{c} x \\ v \end{array} \right] = \left[\begin{array}{c} v \\ -\frac{k}{M}\,x \end{array} \right]$$

$$\frac{d}{dt}\left[\begin{array}{c} x \\ v \end{array} \right] = \left[\begin{array}{cc} 0 & 1 \\ -\frac{k}{M} & 0 \end{array} \right] \left[\begin{array}{c} x \\ v \end{array} \right]$$

qui est de la forme : $\frac{d}{dt}X = A.X$.

3.2.3 Exemple de résolution d'un système

Nous partirons du modèle Prédateur-Proie, qui transcrit une dynamique écologique.

Le premier à proposer ce modèle fut Pierre François Verhulst (1804 - 1849) ; il fut repris ensuite par Alfred J. Lotka(1880-1949) et Vito Volterra(1860-1940). Dans la suite, nous rencontrerons souvent des équatins du type Lotka-Volterra.

Description du problème

Soit 2 populations :

- LES PROIES
  Nous noterons par *x(t)* leur quantité par unité de surface du territoire à l'instant t.
  Elles sont très prolifiques. En l'absence de prédateur, cette population croît exponentiellement.

- LES PRÉDATEURS
  Leur quatité sera notée quant à elle *y(t)*.
  Cette population décroît exponentiellement s'ils ne peuvent se nourrir des proies.

Voici les équations qui gouvernent ces deux populations :

$$\begin{cases} \frac{dx}{dt} &= g.x - p.x.y \\ \frac{dy}{dt} &= -f.y + q.x.y \end{cases}$$

dans lesquelles :
- g est le taux de croissance du nombre de proies ;
- f est le taux de décroissance du nombre de prédateurs ;
- p est le taux de décès des proies ;
- q est le taux de naissance des prédateurs.

Recherche d'une solution

Les variables x et y sont des densités de population. Elles ne peuvent donc être négatifs. Les solutions sont donc uniquement à rechercher pour x et $y > 0$.

Prenons les cas limites :
- si $y(0) = 0$:

$$\begin{cases} x(t) &= x(0).e^{g.t} \\ y(t) &= 0 \end{cases}$$

- si $x(0) = 0$:

$$\begin{cases} x(t) &= 0 \\ y(t) &= y(0).e^{-f.t} \end{cases}$$

Le point d'équilibre est le point tel que $\frac{dx}{dt} = 0$ et $\frac{dy}{dt} = 0$. Donc lorsque $y = \frac{g}{p}$ et $x = \frac{f}{q}$.

Essayons de nous faire une image du système :
- Le long des y : x tend vers 0 ;
- Le long des x : y tend vers infini ;

- 1 point d'équilibre pour une valeur de x et de y ;
- et ailleurs ? Il faut résoudre :

$$\begin{array}{lll} & \frac{dy}{dx} & = \frac{(-f+qx)y}{(g+py)x} \\ \Leftrightarrow & (\frac{g}{y}-p)dy & = (\frac{-f}{x}+q)dx \\ \Leftrightarrow & g.ln(\frac{y}{y_0})+f.ln(\frac{x}{x_0}) & = q(x-x_0)+p(y-y_0) \end{array}$$

Cette dernière équation décrit une courbe fermée dans l'espace des phases (x,y). On dit que la solution est un cycle.

3.3 Motivation de la recherche sur les équations différentielles non linéaires

Le texte suivant, motivant la recherche autour des équations de Lotka-Volterra, est un résumé d'un article[2] basé sur les textes [3, 4, 5, 6, 7].

Nous allons passer au type de systèmes d'équations différentielles qui sera utilisé par la suite : les systèmes autonomes.

Les systèmes autonomes sont des systèmes d'équations différentielles comme présentés ci-dessus, à la différence près qu'aucune variable indépendante n'intervient dans une fonction du membre de droite. Ils sont constitués de la façon suivante :

$$\frac{dx_i(t)}{dt} = P_i(x_1, x_2, ..., x_n); \forall i = 1, .., n \tag{3.2}$$

où $t \in \mathbb{R}, x_i \in \mathbb{R}\, ou\, \mathbb{C}$ et les P_i représentent encore les fonctions f_i des systèmes précédents mais sont plus précisément des polynômes ou, plus généralement, des sommes finies de quasi-monômes du type $x_1^{\alpha_1}. ... .x_n^{\alpha_n}$ avec $\alpha_i \in \mathbb{R}\, ou\, \mathbb{C}$, appelées alors quasi-polynômes.

Remarque : Le préfixe "quasi" n'est présent que si les valeurs ne sont pas entières.

Ces systèmes ne sont pas comme on pourrait le penser de prime abord, plus restrictifs que les précédents. En effet, il suffirait de poser que t est une variable nommée x_{n+1} telle que

$$\frac{dx_{n+1}(t)}{dt} = 0$$

Le système serait alors écrit :

$$\frac{dx_i(t)}{dt} = P_i(x_1, x_2, ..., x_n, x_{n+1}); \forall i = 1, .., n+1$$

avec

$$P_{n+1}(x_1, x_2, ..., x_n, x_{n+1}) = 0$$

Ce système entre bien dans le type de système représenté par 3.2.

La représentation matricielle suivante peut être utilisée pour le système 3.2 :

$$\frac{dx_i(t)}{dt} = x_i \sum_{j=1}^{m} A_{ij} \prod_{k=1}^{n} x_k^{B_{jk}}; \forall i = 1, .., n \tag{3.3}$$

où les A_{ij} et $B_{ij} \in \mathbb{R}\, ou\, \mathbb{C}$ et sont indépendants de t. Le nombre m est lié au nombre de quasi-monômes présents dans le système 3.2, généralement $m \neq n$.

Le facteur x_i en évidence n'est pas anodin : il sert à faire apparaître une dérivée logarithmique de x_i et permet une définition de la matrice B qui lui assure une variance simple sous les transformations suivantes :

$$x_i = \prod_{j=1}^{n} x_j^{'C_{ij}} \tag{3.4}$$

où C est une matrice n x n inversible à éléments dans $\mathbb{R}\, ou\, \mathbb{C}$.

En effet, sous les transformations 3.4, les équations 3.3 restent invariantes de forme, c'est-à-dire que :

$$\frac{dx'_i(t)}{dt} = x'_i \sum_{j=1}^{m} A'_{ij} \prod_{k=1}^{n} x_k^{'B'_{jk}}; \forall i = 1, .., n \tag{3.5}$$

avec les règles suivantes de transformations pour les matrices rectangulaires A (n x m) et B (m x n) :

$$\begin{cases} A' = C^{-1} \circ A \\ B' = B \circ C \end{cases} \tag{3.6}$$

("$\circ$" est le produit matriciel)

Apparaît ainsi la matrice invariante $B \circ A$: il est clair d'après 3.6 que $B' \circ A' = B \circ A$.

Cette matrice m x m va jouer un rôle central dans ce qui suit. Elle est le label des classes d'équivalence en lesquelles la transformation 3.4 scinde l'ensemble des systèmes du type 3.3.

Montrons que cette matrice caractérise une forme canonique des équations de type 3.3, forme canonique particulièrement simple puisque de non-linéarité maximum quadratique. Dans le cas particulier où $m = n$, faisons la transformation 3.4 avec $C = B^{-1}$: la transformation induite 3.6 sur A et B donne donc :

$$\begin{cases} A' = B \circ A \\ B' = I \end{cases}$$

("I" est la matrice identité)

Donc 3.5 s'écrit :

$$\frac{dx'_i(t)}{dt} = \dot{x}'_i = x'_i \sum_{j=1}^{m} (B \circ A)_{ij} x'_j = x'_i \sum_{j=1}^{m} M_{ij} x'_j ; \forall i = 1, .., m \quad (3.7)$$

où la matrice M est définie comme $M = B \circ A$.

C'est de cette équation, et en particulier de cette matrice, dont va traiter le reste de ce mémoire.

Chapitre 4

Connexions affines invariantes par translations associées à une EDO de type Lotka-Volterra

Nous présenterons dans ce chapitre la suite des raisonnements mathématiques et informatiques qui ont été effectués durant toute la recherche pour ce mémoire. Les raisonnements sont détaillés, de manière à ce que le plus grand nombre puisse les comprendre.

Nous commencerons par décrire le calcul du tenseur de Riemann-Christoffel et son implémentation. Nous introduirons les capacités de la méthode avec Riemann.

Nous continuerons par la recherche sur la simplification de matrices par combinaison de changements de coordonnées linéaires et non linéaires. Celle-ci est essentiellement basée sur les idées des chercheurs impliqués dans ce mémoire. Nous étudierons le cas, résolu, de dimension deux et attaquerons le cas de dimension trois. Nous finirons cette recherche par un bref résumé des capacités de ces simplifications.

Enfin, nous introduirons la méthode théorique.

4.1 Calcul du tenseur de Riemann-Christoffel associé à une équation de type Lotka-Volterra de dimension n

Pour une large classe de systèmes d'équations différentielles non linéaires, une équation de Lotka-Volterra est la donnée de ces systèmes. Les deux algorithmes présentés ici permettent d'associer à

un tel système une connexion affine.

4.1.1 Systèmes de Lotka-Volterra

Comme nous l'avons vu en page 23, les systèmes d'équations différentielles non linéaires peuvent être représentés par le système suivant (pour une large classe d'entre eux) :

$$\frac{dx_k(t)}{dt} = x_k \sum_{i=1}^{n} M_{k,i} x_i; (k = 1..n) \tag{4.1}$$

où les coefficients $M_{k,l}$ sont des constantes réelles.

Nous supposons que grâce à *NODES*, nous pouvons passer, sous certaines conditions[1], d'un système d'équations différentielles à une équation de Lotka-Volterra. Nous pouvons donc repartir directement de cette équation. J'ai simplement implémenté la matrice M comme suit :

```
> n := 2 ;
> M := Matrix(n,n,m) ;
```

Matrix est donc définie simplement comme une matrice n x n.

4.1.2 Christoffel

En définissant :

$$\dot{q}^k(t) = x_k(t); (k = 1..n),$$

l'équation 4.1 devient :

$$\ddot{q}^k = \sum_{j,l=1}^{n} \Gamma^k_{j,l} \dot{q}^j \dot{q}^l; (k = 1..n) \tag{4.2}$$

avec

$$\Gamma^k_{j,l} \equiv \frac{1}{2}(M_{l,j}\delta_{k,l} + M_{j,l}\delta_{k,j}); (\forall k, j, l \in [1, .., n]). \tag{4.3}$$

sont les symboles de Christoffel.

Remarque : Dans le cas présent, les $\Gamma^k_{j,l}$ sont des constantes indépendantes des q^k.

Pour la recherche des constantes $\Gamma^k_{j,l}$, j'ai implémenté la fonction *Christoffel,* qui prend comme paramètre la matrice M, et renvoie le tenseur $\Gamma^k_{j,l}$.

[1] Voir Mémoire de Marco Codutti[10], auteur de *NODES*.

Algorithm 1 Christoffel

```
1  : Christoffel := proc(
2  :                          M ## a matrix
                  )
4  :  local TableGamma,n, IDENTITE, k, j, l, f;
5  :  if type(M,matrix) and IsASquareMatrix(M) then
6  :    n := rowdim(M);
7  :    TableGamma := matrix(1,n);
8  :    IDENTITE := Matrix(n,n,shape=identity);
9  :    for k from 1 to n do
10 :      f := (j,l) -> 1/2 * ((M[l,j] * IDENTITE[k,l])
             +(M[j,l] * IDENTITE[k,j]));
11 :      TableGamma[1,k] := matrix(n,n,f)
       od;
13 :    TableGamma;
14 :  else
15 :    error("M is not a square matrix");
      fi :
17 :  end :
```

Explication du code

1 : Déclaration de la procédure

2 : L'unique paramètre de la fonction, la matrice M fournie par l'équation de Lotka-Volterra.

4 : On déclare les variables locales. En *Maple*, chaque variable locale doit être déclarée à la première ligne d'une fonction, avant toute autre instruction.

5 : On vérifie que le paramètre que la fonction reçoit est bien une matrice et que celle-ci est bien carrée. Si ce n'est pas le cas, on se retrouve à la ligne 14.

6 : On récupère la dimension de la matrice M dans la variable n.

7 : On crée une matrice vide de dimensions 1 x n, que l'on nomme TableGamma.

Remarque 1 : *matrix* est une classe de base pour travailler avec les matrices.

Remarque 2 : Pour des raisons de compatibilité de code, nous travaillons toujours avec des matrices plutôt qu'avec des tableaux (*array*) à 1 dimension. Cela permet d'utiliser les mêmes fonctions avec n'importe quel objet créé par la librairie (sauf pour la fonction *Riemann*).

8 : On crée une matrice identité de dimensions n x n (nommée *IDENTITE*) grâce à la classe *Matrix*. Cette classe offre un peu plus de fonctionnalités que *matrix*, comme par exemple de demander que la matrice résultante ait une forme d'une matrice identité ou encore symétrique.

9 : Pour chaque élément de la matrice (k varie de 1 à n)

10 : On crée une application f qui va nous permettre de calculer la valeur désirée pour j et l donnés.

11 : On assigne à l'élément (1,k) de la *TableGamma*, une nouvelle matrice de dimensions n x n, dont les valeurs sont données par l'application f.

13 : On renvoie *TableGamma* à l'utilisateur de la fonction.

15 : On envoie une erreur à l'utilisateur car le paramètre n'est pas une matrice bien formée.

Remarque : Les erreurs et autres textes du programme, destinés à l'utilisateur, sont en langue anglaise car NODES avait pris cette convention.

4.1.3 Riemann

Le tenseur de Riemann, lié à la courbure d'un espace est donné par la formule :

$$R^i_{k,j,l} \equiv \sum_{r=1}^{n} (\Gamma^r_{j,l}\Gamma^i_{k,r} - \Gamma^r_{k,l}\Gamma^i_{j,r}); (\forall i,j,k,l \in [1,..,n])$$

où les $\Gamma^i_{j,k}$ sont données par l'équation 4.3.

J'ai implémenté la recherche des $R^i_{k,j,l}$ par la fonction *Riemann,* qui prend comme paramètre en entrée le tenseur $\Gamma^i_{j,k}$ et renvoie le tenseur $R^i_{k,j,l}$.

Algorithm 2 Riemann

```
1  : Riemann := proc(TableGamma)
2  :   local i,j,k,l,r,R,n ;
3  :   IsAWellformedChristoffelTable(TableGamma) ;
4  :   n := coldim(TableGamma) ;
5  :   R := Array(1..n) ;
6  :   for i from 1 to n do
7  :     R[i] := Array(1..n,1..n,1..n) ;
8  :     for j from 1 to n do
9  :     for k from 1 to n do
10 :     for l from 1 to n do
11 :       R[i][j,k,l] := add(TableGamma[1,r][j,l] * TableGamma[1,i][k,r]
             ,r=1..n) - add(TableGamma[1,r][k,l] *
             TableGamma[1,i][j,r],r=1..n) ;
         od ;od ;od ;
       od ;
14 :   R ;
15 : end :
```

Explication du code

1 : Déclaration de la procédure *Riemann*. Elle prend comme paramètre une table *TableGamma*.

2 : Déclaration des variables locales.

3 : La fonction *IsAWellformedChristoffelTable* permet, comme son nom l'indique, de vérifier si une table correspond à la forme normale d'une table fournie par la fonction *Christoffel*. Cette fonction est décrite dans le chapitre 5, page 51.

4 : On récupère la dimension de cette table dans la variable n.

5 : On crée un vecteur de dimension n que l'on attribue à la variable R. Ce vecteur représente la première dimension (l'indice du haut) dans la formule du calcul de Riemann.

6 : Pour chaque élément du vecteur (i allant de 1 à n)

7 : On crée pour l'élément i du vecteur un tableau à 3 dimensions (n x n x n), qui représente les 3 indices du bas.

8 à 10 : On va parcourir les 3 dimensions du tableau contenu dans l'élément courant du vecteur.

11 : On assigne à l'élément courant du sous-tableau la valeur calculée par la formule décrite ci-dessus.

14 : On renvoie à l'utilisateur le vecteur R.

4.1.4 Recherche des cas d'intégrabilité via le tenseur de Riemann

Des résultats de géométrie différentiells permettent de construire les géodésiques pour les espaces dont le tenseur de Riemann satisfait à l'équation suivante :

$$\nabla R = 0$$

c'est-à-dire

$$\nabla_r R^i_{j,k,l} = \Gamma^i_{s,r} R^s_{j,k,l} - \Gamma^s_{j,r} R^i_{s,k,l} - \Gamma^s_{k,r} R^i_{j,s,l} - \Gamma^s_{l,r} R^i_{j,k,s} = 0$$

dans laquelle on somme sur les indices répétés et $R^i_{j,k,l}$ sont les symboles de Riemann décrits plus haut. Cette équation est nommée la dérivée covariante du tenseur de Riemann.

L'implémentation de cette condition (procédure nommée *DeriveesCovariantesRiemann*) est décrite à la page 57. Les résultats donnés par cet algorithme sont des ensembles d'ensembles de couples $M_{i,j} = val_{i,j}$ qui indique la valeur $val_{i,j}$ que doit prendre l'élément i,j de la matrice M pour que cette technique puisse s'appliquer.

Pour que les résultats soient plus lisibles, une autre fonction a du être écrite afin de transformer ces ensembles de couples en structure matricielles. Elle est nommée *AfficheMatriceAPartirDeCoordonnees* et est décrite page 59.

Pour le cas de dimension 2, voici les résultats que nous donne *Maple*.

$$\begin{bmatrix} m(1,1) & m(1,2) \\ 0 & 0 \end{bmatrix}\begin{bmatrix} m(1,1) & 2\,m(2,2) \\ 0 & m(2,2) \end{bmatrix}$$

$$\begin{bmatrix} m(1,1) & 0 \\ 0 & m(2,2) \end{bmatrix}\begin{bmatrix} m(1,1) & 0 \\ 2\,m(1,1) & m(2,2) \end{bmatrix}$$

$$\begin{bmatrix} 0 & m(1,2) \\ m(2,1) & 0 \end{bmatrix}\begin{bmatrix} 0 & 0 \\ m(2,1) & m(2,2) \end{bmatrix}$$

Les résultats pour la dimension 3 sont mis en annexe (ils sont au nombre de 51).

4.2 Position du problème

Les coefficients $\Gamma_{i,j}^{k}$ apparaissant dans 4.3 peuvent s'interprêter comme les symbôles de Christoffel d'une connexion affine.

Nous rappellons maintenant cette dernière notion.

On considère l'espace $\mathbb{R}^n$ muni de la base canonique e $(=\{e^i\})$.

Définition Une *connexion affine* sur $\mathbb{R}^n$ est une règle qui associe à deux champs de vecteurs $\vec{X}$ et $\vec{v} \in C^\infty(\mathbb{R}^n, \mathbb{R}^n)$, un champ de vecteurs $\nabla_{\vec{v}}\vec{X}$ telle que :

1. $\nabla_{\vec{v}}(a\vec{X}+b\vec{Y}) = a(\nabla_{\vec{v}}\vec{X})+b(\nabla_{\vec{v}}\vec{Y}),\ (\vec{Y} \in C^\infty(\mathbb{R}^n,\mathbb{R}^n); a,b \in \mathbb{R})$
2. $\nabla_{\vec{v}}(f\,X) = (\partial_{\vec{v}}f)(\vec{x})\,\vec{X} + f(\vec{x})\,(\nabla_{\vec{v}}\vec{X}),\ (f \in C^\infty(\mathbb{R}^n))$

   où $\partial_{\vec{v}}f = \sum_{i=1}^{n} v^i \frac{\partial f}{\partial x^i}$
3. $(\nabla_{\alpha\vec{v}+\beta\vec{w}}\vec{X}) = (\alpha\nabla_{\vec{v}}\vec{X})+(\beta\nabla_{\vec{w}}\vec{X}),\ (\alpha,\beta \in C^\infty(\mathbb{R}^n); \vec{w} \in C^\infty(\mathbb{R}^n,\mathbb{R}^n))$

Par exemple :

$$X \longmapsto ((\partial_{\vec{v}}X^1)(\vec{x}),..,(\partial_{\vec{v}}X^n)(\vec{x})) = (\nabla\vec{X})(\vec{x})$$

Dans ce cas :

$$\nabla_{\vec{e_i}}\vec{e_j} = \vec{0}\ \forall i, j.$$

Remarque Si on pose :

$$(\nabla_{\vec{e_i}}\vec{e_j})(\vec{x}) = \Gamma^k_{i,j}\vec{e_k}$$

alors la formule

$$(\nabla_{\vec{v}}\vec{X})(\vec{x}) \quad = \quad v^j(\partial_{\vec{e_j}}X^i)(\vec{x})\vec{e_i} + v^j\Gamma^k_{i,j}X^i\vec{e_{k\vec{x}}}$$

définit une connexion affine sur $\mathbb{R}^n$.

Définition Deux connexions affines ∇^1, ∇^2 sur $\mathbb{R}^n$ sont *équivalentes* si $\exists\varphi : \mathbb{R}^n \to \mathbb{R}^n$ bijective, différentiable, φ^{-1} différentiable telle que

$$\nabla^1_{(d\varphi)(\vec{v})}(d\varphi)(\vec{X}) = (d\varphi)(\nabla^2_{\vec{v}}\vec{X}). \tag{4.4}$$

Proposition Soit ∇ une connexion affine sur $\mathbb{R}^n$ dont les symboles de Christoffel associés sont, dans le système de coordonnées (ξ^i), donnés par les fonctions :

$$\Gamma^k_{i,j}(\vec{\xi}).$$

Soit (x^i) un autre système de cordonnées sur $\mathbb{R}^n$ exprimé en termes de (ξ^i) par la relation

$$\vec{x} = \vec{x}(\vec{\xi})$$

Alors, l'expression des symboles de Christoffel associés à ∇ dans les nouvelles coordonnées (x^i) est donnée par :

$$\bar{\Gamma}^\gamma_{\alpha,\beta}(\vec{x}) = \frac{\partial x^\gamma}{\partial\xi^b}(\vec{\xi})\frac{\partial^2\xi^b}{\partial x^\alpha\partial x^\beta}(\vec{x}) + \frac{\partial\xi^a}{\partial x^\alpha}(\vec{x})\frac{\partial\xi^b}{\partial x^\beta}(\vec{x})\Gamma^c_{a,b}(\vec{\xi})\frac{\partial x^\gamma}{\partial\xi^c}(\vec{\xi}) \tag{4.5}$$

[2]

où l'on somme sur les indices répétés.

Preuve Si on note : $\varphi : \mathbb{R}^n \to \mathbb{R}^n$, le changement de coordonnées

[2]Réf : Doubrovin-Fomenko-Novikov "Géométrie contemporaine" (A ajouter dans la biblio)

$$\vec{\xi} = \varphi(\vec{x}),$$

les symboles de Christoffel de ∇ dans les nouvelles coordonnées (x^i) sont ceux associés à la connexion transportée ∇^φ définie par

$$\nabla^\varphi_{\vec{X}} \vec{Y} = d(\varphi^{-1})(\nabla_{d\varphi(\vec{X})} d\varphi(\vec{Y}))$$

(cf. formule 4.4).
Pour les champs de vecteurs constants $\vec{e}_r$, on a :

$$(d\varphi)_{\vec{x}}(\vec{e}_r) = \frac{\partial \varphi^s}{\partial x^r}(\vec{x})\vec{e_s} = \frac{\partial \xi^s}{\partial x^r}\vec{e}_s,$$

et de même

$$(d\varphi^{-1})_{\vec{\xi}}(\vec{e}_r) = \frac{\partial x^s}{\partial \xi^r}\vec{e_s}.$$

Donc

$$\begin{aligned}
(\nabla^\varphi_{\vec{e}_\alpha}\vec{e}_\beta)(\vec{x}) &= (d\varphi^{-1})_{\varphi(\vec{x})}(\nabla_{d\varphi(\vec{e}_\alpha)} d\varphi(\vec{e}_\beta)) \\
&= (d\varphi^{-1})_{\varphi(\vec{x})}(\nabla_{\frac{\partial \xi^a}{\partial x^\alpha}\vec{e}_a}(\frac{\partial \xi^b}{\partial x^\beta}\vec{e}_b)) \\
&= (d\varphi^{-1})_{\varphi(\vec{x})}(\frac{\partial \xi^a}{\partial x^\alpha}\frac{\partial}{\partial \xi^a}(\frac{\partial \xi^b}{\partial x^\beta})\vec{e}_b + \frac{\partial \xi^a}{\partial x^\beta}\frac{\partial \xi^b}{\partial x^\beta}\Gamma^c_{a,b}\vec{e}_c) \\
(\nabla^\varphi_{\vec{e}_\alpha}\vec{e}_\beta)(\vec{x}) &= \frac{\partial \xi^a}{\partial x^\alpha}\frac{\partial}{\partial \xi^a}(\frac{\partial \xi^b}{\partial x^\beta})\frac{\partial x^\gamma}{\partial}\vec{e}_\gamma + \frac{\partial \xi^a}{\partial x^\alpha}\frac{\partial \xi^b}{\partial x^\beta}\Gamma^c_{a,b}\frac{\partial x^\gamma}{\partial \xi^c}\vec{e}_\gamma \quad (4.6)
\end{aligned}$$

Or

$$\frac{\partial}{\partial \xi^a}(\frac{\partial \xi^b}{\partial x^\beta}) = \frac{\partial}{\partial x^m}(\frac{\partial \xi^b}{\partial x^\beta})\frac{\partial x^m}{\partial \xi^a}$$

Le premier terme de 4.6 s'exprime donc par

$$\begin{aligned}
\frac{\partial \xi^a}{\partial x^\alpha}\frac{\partial x^m}{\partial \xi^a}\frac{\partial}{\partial x^m}(\frac{\partial \xi^b}{\partial x^\beta})\frac{\partial x^\gamma}{\partial \xi^b} &= \frac{\partial x^m}{\partial x^\alpha}\frac{\partial^2(\xi^b)}{\partial x^m \partial x^\beta}\frac{\partial x^\gamma}{\partial \xi^b} \\
&= \delta^m_\alpha \frac{\partial^2 \xi^b}{\partial x^m \partial x^\beta}\frac{\partial x^\gamma}{\partial \xi^b} \\
&= \frac{\partial x^\gamma}{\partial \xi^b}\frac{\partial^2 \xi^b}{\partial x^\alpha \partial x^\beta}
\end{aligned}$$

En remplaçant dans 4.6 on obtient la formule annoncée. CQFD.

Nous pouvons dès à présent énoncer la question qui nous intéresse dans ce chapitre :

Question Soient M et M' deux matrices n x n réelles.
Soient ∇ et ∇' les deux connexions affines respectivement associées à M et M' par les équations 4.1 et 4.2.
Sous quelles conditions M et M' donnent-elles naissance à des connexions ∇ et ∇' équivalentes ?

4.3 Matrices 2 x 2

4.3.1 Changement de coordonnées linéaire

Dans cette section, nous considérons une équivalence linéaire

$$\varphi : \mathbb{R}^n \to \mathbb{R}^n : \vec{x} \mapsto \vec{\xi} \quad = \quad A(\vec{x})$$

où A est une matrice dont le déterminant est différent de 0, car elle devra être inversible dans la suite.

C'est-à-dire :

$$\xi^a = A^a_b x^b$$

et

$$x^\alpha = B^\alpha_\beta \xi^\beta$$

où $B = A^{-1}$(par souci de netteté)

Nous pouvons remarquer que :

$$\frac{\partial x^\alpha}{\partial \xi^m} = B^\alpha_\beta \delta^\beta_m = B^\alpha_m$$

et

$$\frac{\partial^2 \xi^a}{\partial x^\alpha \partial x^\beta} = 0 \tag{4.7}$$

car nous avons admis l'hypothèse de linéarité.

Traduisons la formule 4.5 (page 32) en terme de nos matrices A et B :

$$A^a_\alpha A^b_\beta \Gamma^c_{a,b} B^\gamma_c$$

Le premier terme de l'équation 4.5 n'intervient plus pour la raison énoncé en 4.7.

Travaillons sur l'équation obtenue :

$$\begin{aligned}
\bar{\Gamma}^{\gamma}_{\alpha,\beta} &= A^a_\alpha A^b_\beta \Gamma^c_{a,b} B^\gamma_c \\
&= A^a_\alpha A^b_\beta B^\gamma_c \frac{1}{2}(M_{b,a}\delta_{c,b} + M_{a,b}\delta_{c,a}) \\
&= \frac{1}{2}(A^a_\alpha A^b_\beta B^\gamma_b M_{b,a} + A^a_\alpha A^b_\beta B^\gamma_a M_{a,b}) \\
&= \frac{1}{2}((MA)_{b,\alpha} A^b_\beta B^\gamma_b + A^a_\alpha B^\gamma_a (MA)_{a,\beta}) \\
&= \frac{1}{2}((MA)_{a,\alpha} A^a_\beta B^\gamma_a + A^a_\alpha B^\gamma_a (MA)_{a,\beta}) \\
&= \frac{1}{2} B^\gamma_a ((MA)_{a,\alpha} A^a_\beta + (MA)_{a,\beta} A^a_\alpha) \qquad (4.8)
\end{aligned}$$

Passons à une étude de cas.

1. $A = \kappa I, (\kappa \neq 0)$
   La formule 4.8 devient :

$$\begin{aligned}
&= \frac{1}{2}\frac{1}{\kappa}\delta^\gamma_a(\kappa^2 M_{a,\alpha}\delta^a_\beta + \kappa^2 M_{a,\beta}\delta^a_\alpha) \\
&= \frac{1}{2}(\kappa M_{\gamma,\alpha}\delta^\gamma_\beta + \kappa M_{\gamma,\beta}\delta^\gamma_\alpha).
\end{aligned}$$

   Si nous remplaçons $\kappa M_{\gamma,\alpha}$ par $M'_{\beta,\alpha}$et $\kappa M_{\gamma,\beta}$ par $M'_{\alpha,\beta}$,

$$\frac{1}{2}(M'_{\beta,\alpha}\delta^\gamma_\beta + M'_{\alpha,\beta}\delta^\gamma_\alpha)$$

   nous retrouvons la forme de l'équation 4.3 (page 27).

   Mais peut-on effectuer cette transformation ?
   On peut remarquer que le premier membre de l'équation ne sera pas nul si et seulement si $\gamma = \beta$. Nous pouvons donc remplacer γ par β sans que cela ne pose de problème. Il en est bien sûr de même pour le deuxième membre ($\gamma = \alpha$).

   Conclusion : Les matrices M et κM sont donc équivalente au sens du paragraphe précédent.

   L'un des premiers exercices sur *Maple* était donc d'implémenter ce qui précéde
   La procédure *TableGammaBarreLinear,* décrite à la page 54, permet de trouver la table $\bar{\Gamma}$ à partir de la table Γ et de la matrice de transformation A.

Il restait à trouver la matrice *M'* à partir de la table $\bar{\Gamma}$. Il faut donc inverser le calcul des coefficients de Christoffel.
Pour cela, repartons de l'équation suivante (équation 4.3) :

$$\Gamma^k_{j,l} \equiv \frac{1}{2}(M_{l,j}\delta_{k,l} + M_{j,l}\delta_{k,j}); (\forall k, j, l \in [1, .., n])$$

Remarque : Nous parlerons ici de la matrice M et de la table Γ pour rester dans un cas général.

Extrayons M :

$$M_{l,j}\delta_{k,l} + M_{j,l}\delta_{k,j} \equiv 2\,\Gamma^k_{j,l}$$

On peut donc calculer M de la façon suivante :

Si $k \neq j$ et $k \neq l$: $\Gamma^k_{j,l} = 0$

Si $k = j$ et $k \neq l$: $M_{j,l} = 2\Gamma^j_{j,l}$

Si $k \neq j$ et $k = l$: $M_{l,j} = 2\Gamma^l_{j,l}$

Si $k = j$ et $k = l$: $M_{k,k} = \Gamma^k_{k,k}$

Cet algorithme est implémenté par la procédure *TransformationGammaToMatrix.* Celle-ci est expliquée en page 53.

2. Nous voudrions savoir si un changement de coordonnées (une matrice A) pourrait nous donner *M'* de la forme suivante :

$$M' := \begin{bmatrix} m_{1,1} & K.m_{1,2} \\ m_{2,1} & K.m_{2,2} \end{bmatrix}$$

Comment trouver la matrice de changement de coordonnées ?

Nous connaissons M. Nous pouvons donc calculer la table Γ à partir de M grâce à la fonction *Christoffel*.

Soit A de forme générale :

$$A := \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}$$

Nous pouvons alors calculer la table $\bar{\Gamma}$ en partant de la table Γ et de la matrice A.
Enfin *M'* peut être trouvée grâce à la fonction *Transformation-GammaToMatrix*, précédemment citée.

Nous savons également que la matrice *M'* recherchée est :

$$M'^{vraie} := \begin{bmatrix} m_{1,1} & K.m_{1,2} \\ m_{2,1} & K.m_{2,2} \end{bmatrix}$$

Nous pouvons donc calculer sa table $\bar{\Gamma}$ correspondante, par la fonction *Christoffel*. Nous la nommerons $\bar{\Gamma}^{vraie}$.
Nous avons donc les contraintes suivantes :

$$\forall j, l : M'_{j,l} = (M'^{vraie})_{j,l}$$

et

$$\forall k, j, l : \bar{\Gamma}^k_{j,l} = \left(\bar{\Gamma}^{vraie}\right)^k_{j,l}$$

En cherchant les solutions qui respectent ces contraintes via *Maple*, nous trouvons les formes satisfaisantes pour A.
L'implémentation de cet algorithme nous donne un résultat très élégant :

$$A := \begin{bmatrix} 1 & 0 \\ 0 & K \end{bmatrix}, (K \neq 0)$$

Remarque : pour multiplier par K la première colonne, et non la deuxième, la matrice A devient simplement :

$$A := \begin{bmatrix} K & 0 \\ 0 & 1 \end{bmatrix}$$

Remarque : cet algorithme résout aussi le premier cas énoncé.

Grâce à ce procédé et divisant chaque colonne par un de ses termes, nous pouvons maintenant nous ramener à des matrices d'une des formes :

- $M' := \begin{bmatrix} m'_{1,1} & m'_{1,2} \\ 1 & 1 \end{bmatrix}$
- $M' := \begin{bmatrix} 1 & m'_{1,2} \\ 0 & 1 \end{bmatrix}$
- $M' := \begin{bmatrix} 0 & m'_{1,2} \\ 0 & 1 \end{bmatrix}$

- $M' := \begin{bmatrix} m'_{1,1} & 1 \\ 1 & 0 \end{bmatrix}$
- $M' := \begin{bmatrix} m'_{1,1} & 0 \\ 1 & 0 \end{bmatrix}$
- $M' := \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$
- $M' := \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$
- $M' := \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$
- $M' := \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

4.3.2 Changement de coordonnées quelconque : étude du cas $\begin{bmatrix} m_{1,1} & m_{1,2} \\ 1 & 1 \end{bmatrix}$

Nous voudrions désormais trouver, pour chaque matrice, toutes les matrices qui, par un changement de coordonnées non forcément linéaire, se ramènent à la forme de départ.

Prenons d'abord la matrice de départ :

$$M := \begin{bmatrix} m_{1,1} & m_{1,2} \\ 1 & 1 \end{bmatrix}$$

Etant donné que nous travaillons avec des transformations non linéaires, la procédure *TableGammaBarreLinear* doit être réécrite pour accepter les transformations non linéaires. La procédure qui en résultera sera nommée *TableGammaBarreNonLinear*.

Pour implémenter celle-ci, nous repartons de l'équation 4.5, page 32. Cette équation utilise une table Γ, un système d'équations symbolisant le changement de coordonnées et un ensemble de variables.

Nous pouvons remarquer que dans cet équation, nous devons calculer des dérivées en fonction des x et mais aussi en fonction des ξ, ce qui peut être moins simple à calculer.

Prenons un exemple de système d'équations :

$$\begin{cases} \xi_1 = x_1 x_2 + x_1 \\ \xi_2 = x_1 x_2^3 + x_2 \end{cases}$$

Il est aisé de calculer les dérivées partielles première et seconde de ξ_i par x_j.

Ainsi :

$$(\frac{\partial}{\partial x_1}\xi_1)(x_1, x_2) = x_2 + 1$$

et

$$\frac{\partial^2}{\partial x_1 \partial x_2}\xi_1 = 1$$

Si on inverse la relation $\xi = \xi(x)$ $(x = x(\xi))$, le calcul est moins aisé. Essayons de trouver une astuce afin de simplifier au maximum ce dernier.

$$\delta^\gamma_\alpha = \sum \frac{\partial x^\gamma}{\partial x^\alpha} = \sum_b \frac{\partial x^\gamma}{\partial \xi^b}\frac{\partial \xi^b}{\partial x^\alpha} \tag{4.9}$$

Posons $\frac{\partial \xi^b}{\partial x^\alpha} = \Xi^b_\alpha$(qui est connu) et $\frac{\partial x^\gamma}{\partial \xi^b} = X^\alpha_b$ (que l'on cherche). Alors l'équation 4.9 nous dit que :

$$\Xi X = I \Rightarrow X = \Xi^{-1}$$

Nous avons donc en mains tous les outils pour contruire la procédure *TableGammaBarreNonLinear*. Celle-ci se retrouve à la suite des autres dans le chapitre 5, à la page 56.

Pour arriver à notre but premier, à savoir rechercher toutes les matrices qui, par un changement de coordonnées non forcément linéaire, se ramènent à la forme de M donnée plus haut, nous allons utiliser le même raisonnement que pour le point 2 de l'étude de cas.

Nous avons $M := \begin{bmatrix} m_{1,1} & m_{1,2} \\ 1 & 1 \end{bmatrix}$. Nous pouvons calculer la table Γ correspondante, soit :

$$\Gamma := \begin{bmatrix} \begin{bmatrix} m_{1,1} & \frac{1}{2}m_{1,2} \\ \frac{1}{2}m_{1,2} & 0 \end{bmatrix} & \begin{bmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{bmatrix} \end{bmatrix}$$

Grâce à la fonction *TableGammaBarreNonLinear*, on recherche la table $\bar{\Gamma}$ avec le système d'équations suivant (le plus général possible) :

$$\begin{cases} \xi_1 = f(x_1, x_2) \\ \xi_2 = g(x_1, x_2) \end{cases}$$

On recherche ensuite la matrice M' correspondant à cette table, grâce à la fonction *TransformationGammaToMatrix*.

Maple nous donne les résultats suivants pour *M'* (je ne mentionne ici que les résultats intéressants pour la suite) :

Remarque : Afin de simplifier les sorties, nous utiliserons le raccourcis $\partial_1 f$ pour $\frac{\partial}{\partial x_1} f(x_1, x_2)$. Idem pour x_2 et g.

```
m'(2,1) =
```

$$\frac{-1}{\partial_1 f\,\partial_2 g - \partial_2 f\,\partial_1 g}(2\,\partial_1 g\,\partial_2(\partial_1 f) - 2\,\partial_1 f\,\partial_2(\partial_1 g) + 2\,\partial_1 f\,\partial_2 f\,m_{1,1}\partial_1 g +$$

$$\partial_1 g\,\partial_1 f\,m_{1,2}\partial_2 g + \partial_2 f\,(\partial_1 g)^2 m_{1,2} - \partial_2 g\,(\partial_1 f)^2 - \partial_1 g\,\partial_1 f\,\partial_2 f - 2\,\partial_1 g\,\partial_2 g\,\partial_1 f)$$

```
m'(2,2) =
```

$$\frac{\partial_1 g\,\partial_2(\partial_2 f) - \partial_1 f\,\partial_2(\partial_2 g) + (\partial_2 f)^2 m_{1,1}\partial_1 g + \partial_1 g\,\partial_2 f\,m_{1,2}\,\partial_2 g - \partial_1 f\,\partial_2 g\,\partial_2 f - (\partial_2 g)^2\partial_1 f}{\partial_1 f\,\partial_2 g - \partial_2 f\,\partial_1 g}$$

Nous savons que *M'* doit être de la forme suivante :

$$M'^{vraie} := \begin{bmatrix} m'^{vrai}_{1,1} & m'^{vrai}_{1,2} \\ 1 & 1 \end{bmatrix}$$

Donc $m'_{2,1}$ et $m'_{2,2}$ doivent égaler 1.

Nous allons essayer de nous débarrasser les termes dépendant de la matrice initiale.

Pour cela, la condition $\frac{\partial}{\partial x_1} g(x_1, x_2) = 0$ est suffisante dans les deux termes.

Calculons alors ces termes avec cette condition en plus, via *Maple* grâce à la commande suivante :

```
subs( diff(g(x1,x2),x1)=0,mp(2,1)) ;
```

$$\frac{2\,\partial_1 f\,\partial_2 0 + \partial_2 g\,(\partial_1 f)^2}{\partial_1 f\,\partial_2 g} \tag{4.10}$$

```
subs( diff(g(x1,x2),x1)=0,mp(2,2)) ;
```

$$\frac{\partial_2(\partial_2 g) + \partial_2 g\,\partial_2 f + (\partial_2 g)^2}{\partial_2 g}$$

Remarque : Nous pouvons en effet remarquer que le résultat ne contient plus aucun membre de la matrice initiale

Recherche d'une solution générale pour f

Attardons-nous sur le second résultat. On peut essayer d'en extraire une forme générale pour la fonction f :

$$\begin{aligned} 1 &= \frac{\partial_2(\partial_2 g) + \partial_2 g\, \partial_2 f + (\partial_2 g)^2}{\partial_2 g} \\ \partial_2 f &= 1 - \partial_2 g - \frac{\partial_2(\partial_2 g)}{\partial_2 g} \end{aligned}$$

En utilisant 4.10, nous déduisons :

$$f = x_1 + x_2 - g(x_2) - ln(\partial_2 g)$$

Nous pouvons donc affirmer que le système d'équations suivant

$$\begin{cases} \xi_1 = x_1 + x_2 - g(x_2) - ln(\partial_2 g) \\ \xi_2 = g(x_2) \end{cases}$$

nous permet de passer d'une matrice M telle que

$$M := \begin{bmatrix} m_{1,1} & m_{1,2} \\ 1 & 1 \end{bmatrix}$$

à une matrice M' telle que

$$M' := \begin{bmatrix} m'_{1,1} & m'_{1,2} \\ 1 & 1 \end{bmatrix}$$

qui conserve la forme matricielle initiale.

Lorsque nous passons ces paramètres à *Maple*, la matrice M' trouvée est la suivante :

$$M' := \begin{bmatrix} m_{1,1} & m'_{1,2} \\ 1 & 1 \end{bmatrix}$$

où

$$m'_{1,2} = (ln\, g')'(1 - 2m_{1,1}) + g'(m_{1,2} + 1 - 2m_{1,1}) - (1 - 2m_{1,1}) \quad (4.11)$$

En posant $g = cx_2 (c > 0)$, l'expression de $m'_{1,2}$ devient

$$m'_{1,2} = c(m_{1,2} + 1 - 2m_{1,1}) - (1 - 2m_{1,1})$$

Ce qui s'annule pour

$$c = \frac{1 - 2m_{1,1}}{m_{1,2} + 1 - 2m_{1,1}},$$

pour vu que

$$(a) \begin{cases} 1 - 2m_{1,1} > 0 \\ m_{12} + 1 - 2m_{1,1} > 0 \end{cases},$$

ou

$$(b) \begin{cases} 1 - 2m_{1,1} < 0 \\ m_{12} + 1 - 2m_{1,1} < 0 \end{cases}.$$

C'est-à-dire

$$(a) \begin{cases} m_{1,1} < \frac{1}{2} \\ m_{12} > 2m_{1,1} - 1 \end{cases},$$

ou

$$(b) \begin{cases} m_{1,1} > \frac{1}{2} \\ m_{12} < 2m_{1,1} - 1 \end{cases}.$$

Le cas $m_{1,1} = \frac{1}{2}$ se traite de manière similaire. On a $m'_{1,2} = g' m_{1,2}$. On remène donc, par $g = cx_2 (c > 0)$ le coefficient (12) à

$$(c)\, m'_{1,2} = \pm 1$$

(ou 0 dans le cas $m_{1,2} = 0$).

On obtient donc le résultat suivant.

Proposition Soit $M = \begin{bmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{bmatrix}$ une matrice 2 x 2 réelle satisfaisant une des conditions suivantes :

- (a) $2m_{1,1} < m_{2,1}$ et $det\, M < m_{2,2}(m_{2,1} - m_{1,1})$
- (b) $2m_{1,1} > m_{2,1}$ et $det\, M > m_{2,2}(m_{2,1} - m_{1,1})$
- (c) $2m_{1,1} = m_{2,1}$.

Alors, la connexion affine associée sur $\mathbb{R}^2$ est équivalente à celle associée à une matrice M' de la forme

1. $M' = \begin{bmatrix} m & 0 \\ 1 & 1 \end{bmatrix}$ dans les premiers cas ;
2. $M' = \begin{bmatrix} m & E \\ 1 & 1 \end{bmatrix}$ avec $E = \pm 1, 0$ dans le dernier cas.

Nous présentons maintenant un résultat local (valide sur des ouverts propres de $\mathbb{R}^2$ seulement) relatif aux complémentaires des conditions (a),(b) et (c).

On cherche comme précédemment à annuler $m'_{1,2}$, c'est-à-dire à résoudre l'équation différentielle :

$$(lng')'a + g'b - a = 0$$

où $a := 1 - 2m_{1,1}$ et $b := m_{1,2} + a$.

En supposant $a \neq 0$ (i.e. $m_{1,1} \neq \frac{1}{2}$) et en posant $u = g'$, on obtient

$$u' = u - \frac{b}{a}u^2.$$

Il s'agit d'une équation à variables séparées dont la solution est

$$x_2 = \int \frac{du}{u(1 - \frac{b}{a}u)}.$$

La primitive se calcule explicitement par une décomposition en fractions simples. On obtient formellement

$$x_2 + C = ln(\frac{u}{a - bu}),$$

ou encore

$$Ke^{x_2} = \frac{u}{a - bu} \, (K > 0).$$

De là on tire

$$u = g' = \frac{aKe^{x_2}}{bKe^{x_2} + 1},$$

d'où

$$\begin{cases} g = C + \frac{a}{b}ln(bKe^{x_2} + 1) & (b \neq 0) \\ g = aKe^{x_2} + C & (b = 0) \end{cases}$$

avec $K > 0$ et $C \in \mathbb{R}$ arbitraires.

Dans les deux cas ($b = 0$ et $b \neq 0$) l'application $\varphi = (f, g)$ n'est pas un difféomorphi(?) de $\mathbb{R}^2$. L'annulation $m'_{1,2} \equiv 0$ est donc locale uniquement.

Comme on peut le voir, il suffit de définir le terme $m'_{1,2}$ voulu pour qu'il apparaisse dans la matrice finale.

Cette technique est donc très utile pour travailler avec des matrices beaucoup plus simples (plus qu'une seule variable : $m_{1,1}$, plutôt que les quatre variables initiales).

Deuxième matrice étudiée

De la même manière, nous pouvons étudier la matrice M suivante :

$$M := \begin{bmatrix} 1 & 1 \\ m_{2,1} & m_{2,2} \end{bmatrix}$$

Les calculs sont sensiblement les mêmes. Il suffit de suivre les mêmes étapes que pour la matrice précédente. On remarque alors que tous les calculs restent valables, avec les substitutions suivantes :

$$\begin{aligned} f &\Rightarrow g \\ g &\Rightarrow f \\ x_1 &\Rightarrow x_2 \\ x_2 &\Rightarrow x_1 \\ m_{1,1} &\Rightarrow m_{2,2} \\ m_{1,2} &\Rightarrow m_{2,1} \end{aligned}$$

Troisième et quatrième matrices étudiées

Partons des deux dernières matrices à étudier, c'est-à-dire :

$$M := \begin{bmatrix} m_{1,1} & 1 \\ 1 & m_{2,2} \end{bmatrix}$$

$$M := \begin{bmatrix} 1 & m_{1,2} \\ m_{2,1} & 1 \end{bmatrix}$$

Nous étudierons la première matrice. Comme précédemment, on voit aisément que les résultats seront symétriques pour la seconde.

Si nous utilisons la fonction *TableGammaBarreNonLinear* avec les équations générales suivantes :

$$\begin{cases} f = f(x_1, x_2) \\ g = g(x_1, x_2) \end{cases}$$

nous obtenons une table *TableGammaBarre* qui provient d'une matrice *M'* que l'on peut retrouver grâce à la fonction *TransformationGammaToMatrix*.

De nouveau, les termes [1,2] et [2,1] de *M'* doivent être égaux à 1 si on veut que les formes de M et *M'* soient identiques.

Comme précédemment, nous voulons que ces 2 termes soient indépendants de la matrice initiale M. La condition n'est plus aussi simple que pour les deux cas précédents. Elle se présente comme suit :

$$(d_1 f = 0 \text{ et } d_2 g = 0) \text{ ou } (d_2 f = 0 \text{ et } d_1 g = 0)$$

Etudions les deux conditions :

1. $d_2 f = 0$ et $d_1 g = 0$
   Les équations sont donc de la forme :

$$\begin{cases} f = f(x_1) \\ g = g(x_2) \end{cases}$$

   Nous repassons la matrice initiale par la fonction *TableGamma-BarreNonLinear* avec ces équations. Nous obtenons une nouvelle matrice *M'*.
   Une nouvelle fois, les termes [1,2] et [2,1] doivent être égaux à 1 pour les mêmes raisons que plus haut :

$$\begin{cases} d_1 f = 1 \\ d_2 g = 1 \end{cases}$$

   Les équations prennent donc la forme suivante :

$$\begin{cases} f = x_1 + C_f \\ g = x_2 + C_g \end{cases}$$

   Ces équations permettent de passer d'une matrice M à une matrice *M'*, identique à la matrice M initiale, ce qui ne nous avance pas beaucoup.
2. $d_1 f = 0$ et $d_2 g = 0$
   Les équations sont donc de la forme :

$$\begin{cases} f = f(x_2) \\ g = g(x_1) \end{cases}$$

   Nous faisons repasser la matrice initiale par la fonction *Table-GammaBarreNonLinear* avec ces équations. Nous obtenons une nouvelle matrice *M'*.
   Les termes [1,2] et [2,1] doivent être égaux à 1 :

$$\begin{cases} d_2 f = 1 \\ d_1 g = 1 \end{cases}$$

   Les équations ont donc la forme suivante :

$$\begin{cases} f = x_2 + C_f \\ g = x_1 + C_g \end{cases}$$

   Ces équations permettent de passer d'une matrice M à une matrice *M'* telle que :

$$M' = \begin{bmatrix} m_{2,2} & 1 \\ 1 & m_{1,1} \end{bmatrix}$$

En plus de garder la forme générale (les deux 1), ces équations permettent d'échanger les termes $m_{1,1}$et $m_{1,2}$, ce qui n'est pas directement utile dans le cadre de ce mémoire.

4.3.3 Méthode théorique

Nous venons de voir les résultats pour la méthode expérimentale que nous expections.

La méthode théorique peut nous donner aussi des résultats, en particulier pour les matrices 2 x 2. Il serait intéressant de les comparer. Mais avant cela, voyons jusqu'où elle nous amène.

Grâce à la méthode théorique, nous pouvons rechercher la solution générale de :

$$\begin{cases} \frac{dx_1}{dt} = x_1(M_{11}x_1 + M_{12}x_2) \\ \frac{dx_2}{dt} = x_2(M_{21}x_1 + M_{22}x_2) \end{cases}$$

autrement dit :

$$\frac{dx_i}{dt} = x_i \sum_{j=1}^{2} M_{ij}x_j; \forall\, i = 1, 2 \tag{4.12}$$

Si la matrice M est inversible (c'est-à-dire que le déterminant de M est différent de 0), on peut faire la transformation quit suit :

$$\frac{dx'_i}{dt} = x'_i \prod_{j=1}^{2} x_j^{'M_{ij}}; \forall\, i = 1, 2 \tag{4.13}$$

Preuve

L'équation 4.12 peut être vue comme :

$$\frac{dx_i}{dt} = x_i \sum_{j=1}^{m} A_{ij} \prod_{k=1}^{n} x_k^{B_{jk}}; \forall i = 1, 2 \tag{4.14}$$

dans laquelle $A_{ij} = M_{ij}$ et $B_{jk} = \delta_{jk}(\delta_{jk} = 1$ si $j = k$, 0 sinon).

De plus, nous pouvons ramener l'équation 4.14 vers 3.5 (page 24) :

$$\frac{dx'_i}{dt} = x'_i \sum_{j=1}^{m} A'_{ij} \prod_{k=1}^{n} x_k^{'B'_{jk}}; \forall i = 1, 2 \tag{4.15}$$

avec

$$\begin{cases} A' = C^{-1} \circ A = M^{-1} \circ M = I \\ B' = B \circ C = I.M = M \end{cases}$$

(“I “ représente la matrice indentité)
L'équation 4.15 se réduit donc à :

$$\frac{dx'_i}{dt} = x'_i \prod_{j=1}^{2} x_j^{'M_{ij}}; \forall\, i = 1, 2$$

qui est l'équation recherchée 4.13.

Décomposons cette équation :

$$\begin{aligned} \frac{dx'_1}{dt} &= x'_1 x_1^{'M_{11}} x_2^{'M_{12}} \\ \frac{dx'_2}{dt} &= x'_1 x_1^{'M_{11}} x_2^{'M_{12}} \end{aligned} \tag{4.16}$$

$$\frac{dx'_1}{x'_1 x_1^{'M_{11}} x_2^{'M_{12}}} = dt = \frac{dx'_2}{x'_2 x_1^{'M_{21}} x_2^{'M_{22}}}$$

Il est possible de séparer les variables de cette équation :

$$\frac{dx'_1}{x_1^{'M_{11}+1} x_2^{'M_{12}}} = \frac{dx'_2}{x_1^{'M_{21}} x_2^{'M_{22}+1}}$$

$$\Leftrightarrow dx'_1 x_1^{'M_{21}-M_{11}-1} = dx'_2 x_2^{'M_{12}-M_{22}-1}$$

Nous pouvons intégrer les deux membres de l'équation

$$\Leftrightarrow \int dx'_1 x_1^{'M_{21}-M_{11}-1} = \int dx'_2 x_2^{'M_{12}-M_{22}-1} + C$$

dans laquelle C est une constante.
Or, il est connu que

$$\int dx\, x^a = \frac{x^{a+1}}{a+1}$$

Alors :

$$\Leftrightarrow \frac{x_1^{'M_{21}-M_{11}}}{M_{21}-M_{11}} = \frac{x_2^{'M_{12}-M_{22}}}{M_{12}-M_{22}} + C$$

Exprimons x'_2 en terme de x'_1 et des éléments de M.

$$x'_2 = \left\{ \left[-C + \frac{x_1^{'M_{21}-M_{11}}}{M_{21}-M_{11}} \right] (M_{12}-M_{22}) \right\}^{\frac{1}{M_{12}-M_{22}}}$$

Nous pouvons remplacer, dans la première équation du système 4.16, x'_2 par cette valeur. Nous obtenons :

$$\begin{aligned}
\frac{dx'_1}{dt} &= x_1'^{M_{11}+1} \left\{ \left[-C + \frac{x_1'^{M_{21}-M_{11}}}{M_{21}-M_{11}} \right] (M_{12}-M_{22}) \right\}^{\frac{M_{12}}{M_{12}-M_{22}}} \\
dt &= dx'_1 x_1'^{M_{11}+1} \left\{ \frac{M_{12}-M_{22}}{M_{21}-M_{11}} x_1'^{M_{21}-M_{11}} + C' \right\}^{\frac{M_{12}}{M_{22}-M_{12}}} \\
t - t_0 &= \int_{x'_1(t_0)}^{x'_1(t)} dx'_1 x_1'^{M_{11}+1} \left\{ \frac{M_{12}-M_{22}}{M_{21}-M_{11}} x_1'^{M_{21}-M_{11}} + C' \right\}^{\frac{M_{12}}{M_{22}-M_{12}}}
\end{aligned}$$

Nous avons donc une solution générale pour le cas de dimension 2. Il est à noter que cette équation est intégrable mais qu'elle n'est pas toujours exprimable en terme de fonctions élémentaires. Malheureusement, nous n'avons pas pu aller plus loin par faute de temps dans la comparaison de cette méthode avec la nôtre.

4.4 Matrices 3 x 3

Maintenant que nous avons exploré les matrices carrées de dimension 2, nous pouvons commencer celles de dimension supérieure.

Nous allons reprendre le canevas précédent (recherche de transformations linéaires puis non linéaires) pour essayer de simplifier au maximum des matrices de dimension 3, par des changements de coordonnées.

Nous pouvons dès à présent faire certaines remarques :

1. Tandis qu'un matrice 2 x 2 n'a que 4 variables au maximum, une matrice 3 x 3 en possède 9. La compléxité des calculs s'en ressentira directement.
2. Le changement de coordonnées linéaire permettant (dans le cas de dimension 2) de multiplier par un nombre une colonne choisie :

   $$A := \begin{bmatrix} k & 0 \\ 0 & 1 \end{bmatrix}$$

   est généralisable dans le cas de la dimension 3.
   En effet, la matrice de transformation

   $$A := \begin{bmatrix} k & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

   permet de multiplier la première colonne d'une matrice 3 x 3 par k. Nous pouvons donc déjà éliminer 3 variables sur 9.
   Le nombre de matrices que l'on peut trouver grâce à cette transformation s'élève à 27 (3 places possibles pour le 1 dans la première colonne, 3 dans la deuxième et 3 dans la dernière : 3 x 3 x 3 = 27)
3. Vous conviendrez aisément qu'étudier 27 matrices différentes sous tous les angles, peut être un travail lourd et pénible. Il est alors dans notre intérêt de trouver un moyen de réduire ces matrices à un échantillon plus petit mais qui représente encore la totalité de ces matrices.
   L'idée est la suivante : si nous pouvons transformer une matrice contenant un 1 dans chaque colonne vers une autre matrice ayant cette même propriété, nous pouvons omettre d'étudier la première.
   En effet, si une matrice ayant la forme de la première est rencontrée, il suffit de lui appliquer la transformation pour travailler

avec la matrice sous la seconde forme.
Il n'y a plus qu'à traduire cela en un algorithme pour *Maple*. Nous le nommerons *ReductionEnsembleMatrices* et est décrite à la page 61.
Les matrices de transformations linéaires possibles, ne contenant que des 1, sont les suivantes :

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, A := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, A := \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$A := \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, A := \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \text{ et } A := \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

Nous savons que la première ne permettra pas de réduire l'ensemble car elle envoie chaque matrice sur elle-même (A est la matrice identité).
Les autres transformations permettent tour à tour de raffiner l'ensemble pour arriver à un échantillon de 7 matrices :

$$\begin{bmatrix} 1 & . & . \\ . & 1 & . \\ . & . & 1 \end{bmatrix}, \begin{bmatrix} . & . & . \\ . & 1 & . \\ 1 & . & 1 \end{bmatrix}, \begin{bmatrix} . & . & . \\ . & . & . \\ 1 & 1 & 1 \end{bmatrix}, \begin{bmatrix} . & . & 1 \\ 1 & . & . \\ . & 1 & . \end{bmatrix}, \begin{bmatrix} 1 & . & 1 \\ . & . & . \\ . & 1 & . \end{bmatrix},$$

$$\begin{bmatrix} . & 1 & . \\ 1 & . & . \\ . & . & 1 \end{bmatrix}, \begin{bmatrix} . & . & . \\ . & . & 1 \\ 1 & 1 & . \end{bmatrix}$$

Pour résumer, à partir de ces 7 matrices et par combinaison partielle des 6 transformations linéaires, nous pouvons retrouver les 27 matrices initiales.

L'étude des matrices de dimensions 3 x 3 se termine ici. Sans automatisation du processus de recherche, il est très difficile d'avancer rapidemment.

Chapitre 5

Algorithmes

Nous allons présenter ici tous les algorithmes dont nous avons parlé dans le chapitre précédent. Chaque algorithme est introduit par une brève description, puis énoncé et enfin commenté pour comprendre chaque partie du code.

De plus, durant le processus de la recherche de solutions, certaines fonctions supplémentaires ont aidé à la réalisation de ce que demandaient les chercheurs. Elles trouvent donc naturellement leur place dans ce chapitre.

La lecture de ce chapitre peut être intéressante dans le but de comprendre totalement le code fourni avec ce mémoire mais peut être omise pour une première lecture.

5.1 IsASquareMatrix

Voici une fonction très simple, mais ô combien ! utile, qui permet de savoir si une matrice est carrée ou pas.

Algorithm 3 IsASquareMatrix

```
1  : IsASquareMatrix := proc (
2  :                           M ## a matrix
                              )
4  : if rowdim(M) = coldim(M) then true else false fi :
     end :
```

5.2 IsAWellFormedChristoffelTable

Il est bon de savoir si les objets que l'on manipule possèdent les propriétés que l'on est en droit d'attendre de ceux-ci. Dans cette optique, cette fonction renvoie une erreur (et donc arrête le programme en cours d'éxecution) si une propriété n'est pas respectée.

Les propriétés

Pour trouver les propriétés nécessaires pour qu'une table soit une table de Christoffel, nous pouvons regarder comment la table est construite par la fonction *Christoffel*.

- La table doit être une matrice à 2 dimensions 1 x n ;
- Chaque élément de cette matrice doit être une matrice de dimensions n x n.

Algorithm 4 IsAWellFormedChristoffelTable

```
1  : IsAWellFormedChristoffelTable := proc (
2  :                                    TableGamma
                                       )
4  : local n,i;
5  : if type(TableGamma, matrix) then
6  :   if rowdim(TableGamma) = 1 then
7  :     n := coldim(TableGamma);
8  :     for i from 1 to n do
9  :       if not type(TableGamma[1,i],matrix) or
           not IsASquareMatrix(TableGamma[1,i]) or
           not (coldim(TableGamma[1,i]) = n) then
10 :         error("TableGamma isn't wellformed matrix");
         fi;
       od;
13 :   else
14 :     error("TableGamma must have only one row");
       fi :
16 : else
17 :   error("TableGamma isn't a matrix");
     fi :
     end :
```

Explication du code

1 : Déclaration de la fonction.

2 : Celle-ci prend comme paramètre une table TableGamma.

4 : Déclaration des variables locales.

5 : On vérifie que la table passée en paramètre est bien une matrice.

6 : Celle-ci ne doit posséder qu'une seule ligne.

7 : On récupère le nombre de colonnes dans la variable n.

8 : On va parcourir les éléments de la matrice (i variant de 1 à n).

9 : Pour chaque élément, on vérifie qu'il est bien aussi une matrice, que celle-ci est carrée dont les dimensions sont bien n x n.

10 : Si ce n'est pas le cas (une condition n'est pas vérifiée), on envoie une erreur.

13 - 14 : Envoi d'un message d'erreur si la table possède plus d'une ligne.

16 - 17 : Envoi d'un message d'erreur si la table n'est pas une matrice.

5.3 TransformationGammaToMatrix

Cette fonction permet de retrouver une matrice de dimensions n x n qui est la matrice initiale d'une table de Christoffel de dimension n. Elle se base sur les propriétés définies en page 36.

Rappel

Pour *j,k,l* donnés :

Si $k \neq j$ et $k \neq l$: $\Gamma^k_{j,l} = 0$

Si $k = j$ et $k \neq l$: $M_{j,l} = 2\Gamma^j_{j,l}$

Si $k \neq j$ et $k = l$: $M_{l,j} = 2\Gamma^l_{j,l}$

Si $k = j$ et $k = l$: $M_{k,k} = \Gamma^k_{k,k}$

Algorithme

Algorithm 5 TransformationGammaToMatrix

```
1  : TransformationGammaToMatrix := proc(
2  :         TableGamma ## la table des gamma donnée par
Christoffel
                                          )
4  : local n,M,k,j,l;
5  : IsAWellformedChristoffelTable(TableGamma);
6  : n := coldim(TableGamma);
7  : M := matrix(n,n);
8  : for j from 1 by 1 to n do
9  : for l from 1 by 1 to n do
10 :   if j = l then
11 :     M[j,j] := TableGamma[1,j][j,j];
12 :   else
13 :     M[l,j] := 2 * TableGamma[1,l][j,l];
14 :   end if;
15 : end do;
16 : end do;
17 : M;
     end :
```

Explication du code

1 : Déclaration de la procédure.

2 : *TableGamma* est le seul paramètre. Il représente la table de Christoffel à partir de laquelle la matrice initiale doit être retrouvée.

4 : Déclaration des variables locales.

5 : On vérifie que la table donnée en paramètre possède les propriétés d'une table de Christoffel, grâce à la fonction *IsAWellFormedChristoffelTable*.

6 : On récupère la dimension de la table dans la variable n.

7 : On crée la matrice M de dimensions n x n initialement vide.

8-9 : On va parcourir tous les éléments de M (j et l allant de 1 à n)

10-14 : On applique les formules définies plus haut pour l'élément (j,l).

17 : On renvoie la matrice M construite.

5.4 TableGammaBarreLinear

Cette fonction permet de passer d'une table *TableGammaXi* vers une table *TableGammaBarreLinear* par un changement de coordonnées linéaire définie par la matrice *MatrixA*.

Algorithm 6 TableGammaBarreLinear

```
1  : TableGammaBarreLinear := proc(
2  :                MatrixA, ## Coordinates Transformation matrix
3  :                TableGammaXi
                                )
5  : local n,MatrixAInverse,TableGammaBarreLinear, k, j, l, f;
6  : IsAWellformedChristoffelTable(TableGammaXi);
7  : n := coldim(TableGammaXi);
     ## Verification des invariants
9  : if (not IsASquareMatrix(MatrixA)) then
10 :   error "MatrixA must be a squarematrix";
11 : fi;
12 : if not (rowdim(MatrixA) = n) then
13 :   error "MatrixA hasn't same order that TableGammaXi";
14 : fi;
     ## inversion de la matrice A
16 : MatrixAInverse := inverse(MatrixA);
     ## Création de la table GammaBarre
18 : TableGammaBarreLinear := matrix(1,n);
19 : for k from 1 to n do
20 : f := (j,l) -> add(add(add(MatrixA[a,j] * MatrixA[b,l] *
                   TableGammaXi[1,c][a,b] *
                   MatrixAInverse[k,c],a=1..n),b=1..n),c=1..n);
21 : TableGammaBarreLinear[1,k] := matrix(n,n,f);
22 : od;
23 : TableGammaBarreLinear;
     end :
```

Explication du code

1 : Déclaration de la procédure

2-3 : La fonction prend 2 paramètres : *MatrixA*, la matrice qui symbolise le changement de coordonnées et *TableGammaXi*, une table fournie par la fonction *Christoffel*.

5 : Déclaration des variables locales.

6 : On vérifie que la table fournie possède les bonnes propriétés.

7 : On récupère le nombre de colonnes de cette table dans la variable n. Nous sommes sûrs que son nombre de lignes vaut 1.

9-14 : *MatrixA* doit être carrée et son nombre de lignes (ou de colonnes) doit être égal à n.

16 : On inverse la matrice *MatrixA* via la commade *inverse* de la librairie *linalg*. Le résultat est attribué à la variable *MatrixAInverse*.

18 : On crée une matrice vide de dimensions 1 x n dans la variable *TableGammaBarreLinear*.

19 : Pour chaque élément de cette matrice (k variant de 1 à n)

20 : On construit une application f qui attribue une valeur pour j et l donnés.

21 : On attribue à l'élément (1,k) de *TableGammaBarreLinear* une matrice de dimensions n x n dont les valeurs sont calculées grâce à l'application f.

23 : On renvoie à l'utilisateur la table *TableGammaBarreLinear* construite.

5.5 TableGammaBarreNonLinear

Cette fonction permet la même opération que la précédente, mais avec un changement de coordonnées non linéaire. A la place d'une matrice de transformation, nous avons maintenant un ensemble d'équations (qui peuvent être linéaires ou non) et un ensemble de variables qui y interviennent.

Algorithm 7 TableGammaBarreNonLinear

```
1  : TableGammaBarreNonLinear := proc(
2  :                     Equations, ## tableau qui exprime les
équations en terme de xi1,xi2,... et variables
3  :                     variables, ## les variables de
dérivations
4  :                     TableGammaXi ## une table fourni par la
fonction Christoffel
                                 )
6  : local n,k,f,MatD,MatInvD,TableGammaBarre;
7  : IsAWellformedChristoffelTable(TableGammaXi);
8  : n := coldim(TableGammaXi);
9  : f := (j,l) -> diff(Equations[j],variables[l]);
10 : MatD := matrix(n,n,f);
11 : MatInvD := inverse(MatD);
12 : TableGammaBarre := matrix(1,n);
13 : for k from 1 to n do
14 :   f := (j,l) -> add(MatInvD[k,d]*diff(MatD[d,j],variables[l]),d=1..n)
+ add(add(add((MatD[a,j]*MatD[b,l]*(TableGammaXi[1,c][a,b])*MatInvD[k,c])
,a=1..n),b=1..n),c=1..n);
15 :   TableGammaBarre[1,k] := matrix(n,n,f);
16 : od;
17 : TableGammaBarre;
     end :
```

Explication du code

1 : Déclaration de la procédure

2 : Nous avons comme premier paramètre l'ensemble d'équations symbolisant le changement de coordonnées. Celui-ci est noté *Equations*.

3 : Le second paramètre, *variables*, est simplement l'ensemble des variables intervenant dans *Equations*.

4 : *TableGammaXi* est le troisième et dernier paramètre. Il représente les symboles de Christoffel de la matrice que l'on veut étudier. Ceux-ci sont fournis par la fonction *Christoffel*.

6 : Nous déclarons les variables locales.

7 : Nous vérifions que le paramètre *TableGammaXi* correspond bien à une table que pourrait fournir la fonction *Christoffel* via la fonction *IsAWellFormedChristoffelTable*, décrite plus haut dans ce chapitre.

8 : Nous récupérons la dimension n de la table *TableGammaXi*.

9 : Nous créons une application f qui va attribuer à un élément (j,l), la dérivée partielle de l'équation j par rapport à la variable l.

10 : Nous créons ensuite la matrice des dérivées grâce à l'application f que nous attribuons à la variable *MatD*. L'élément (j,l) de la matrice représente donc la dérivée partielle de l'équation j par rapport à la variable l.

11 : Nous attribuons à *MatInvD* l'inverse de la matrice *MatD*, calculée par la fonction *inverse* de la librairie *linalg*.

12 : *TableGammaBarre* est un vecteur de dimension n. Pour une compatibilité entre les objets, nous la définissons plutôt comme une matrice comprenant qu'une seule ligne et n colonne.

13 : Pour chaque élément de cette matrice :

14 : On redéfinie l'application f comme étant l'équation 4.5 (page 32) pour les j,k,l courants.

15 : Nous attribuons à l'élément courant de *TableGammaBarre* une matrice de dimensions n x n, dont les éléments sont calculés à partir de l'application f.

17 : La table *TableGammaBarre* est renvoyée à l'utilisateur.

5.6 DeriveesCovariantesRiemann

Nous avons vu à la page 30 que grâce à la fonction *DeriveesCovariantesRiemann*, nous pouvions retrouver toutes les matrices des équations de Lotka-Volterra que le procédé via Riemann peut résoudre.

Cette procédure implémente la condition

$$\nabla_r R^i_{j,k,l} = \Gamma^i_{s,r} R^s_{j,k,l} - \Gamma^s_{j,r} R^i_{s,k,l} - \Gamma^s_{k,r} R^i_{j,s,l} - \Gamma^s_{l,r} R^i_{j,k,s} = 0$$

puis la recherche de solutions pour celle-ci.

Les résultats sont affichés grâce à la fonction *AfficheMatriceAPartirDeCoordonnees* décrite plus loin dans ce chapitre.

Algorithm 8 DeriveesCovariantesRiemann

```
1  : DeriveesCovariantesRiemann := proc(n)
2  :   local R,i,j,k,l,s,r,Contraintes,Solutions,TR,NR,TableGamma,
variables;
3  :   Contraintes := {};
4  :   M := matrix(n,n,m);
5  :   TableGamma := Christoffel(M);
6  :   TR := Riemann(TableGamma);
7  :   NR := Array(1..n);
8  :   for r from 1 to n do
9  :     NR[r] := Array(1..n);
10 :     for i from 1 to n do
11 :       NR[r][i] := Array(1..n,1..n,1..n);
12 :       for j from 1 to n do
13 :       for k from 1 to n do
14 :       for l from 1 to n do
15 :         NR[r][i][j,k,l] := add(TableGamma[1,i][s,r] *
TR[s][j,k,l],s=1..n)
                              - add(TableGamma[1,s][j,r] *
TR[i][s,k,l],s=1..n)
                              - add(TableGamma[1,s][k,r] *
TR[i][j,s,l],s=1..n)
                              - add(TableGamma[1,s][l,r] *
TR[i][j,k,s],s=1..n);
16 :         Contraintes := {op(Contraintes), NR[r][i][j,k,l] =
0};
17 :       od;od;od;
18 :     od;
19 :   od;
20 :   printf("Les contraintes :");
21 :   print(Contraintes);
22 :   variables := {};
23 :   for i from 1 to n do
24 :   for j from 1 to n do
25 :     variables := {op(variables),eval(M[i,j])};
26 :   od;od;
27 :   printf("Les variables :");
28 :   print(variables);
29 :   Solutions := {solve(Contraintes,variables)};
30 :   printf("Les solutions(%d) : ",nops(Solutions));
31 :   print(Solutions);
32 :   AfficheMatriceAPartirDeCoordonnees(n,Solutions);
33 : end :
```

Explication de l'algorithme

1 : Déclaration de la procédure. Un paramètre : n, symbolisant la dimension dans laquelle on va travailler.

2 : Déclaration des variables locales que l'on va utiliser dans cet

algorithme.

3 : On initialise l'ensemble des contraintes à vide.

4 : On crée une matrice M, de dimensions n x n.

5 : On récupère la table des Γ donné par la fonction *Christoffel* dans la variable *TableGamma*.

6 : On calcule le tenseur de Riemann à partir de *TableGamma* qu'on attribue à la variable *TR*.

7-14 : La variable *NR* va contenir les différents éléments de la condition. Elle aura comme structure un ensemble tableaux encastrés. Ces dimensions sont $[n][n][n \text{ x } n \text{ x } n]$. Ces lignes permettent de créer cette structure.

15 : On calcule la condition pour r,i,j,k et l donnés.

16 : On ajoute la condition courante à l'ensemble des contraintes *Contraintes*.

20-21 : On affiche l'ensemble des contraintes finales.

22 : On initialise l'ensemble des variables à vide.

23-26 : On ajoute à cet ensemble tous les éléments de la matrice initiale M.

27-28 : On affiche cet ensemble de variables.

29 : On recherche les solutions possibles pour les variables données suivant les contraintes définies par l'ensemble *Contraintes*.

30-31 : On imprime les solutions trouvées (ainsi que son nombre)

32 : On affiche ces solutions sous forme de matrices pour que ce soit plus visuel via la fonction *AfficheMatriceAPartirDeCoordonnees*.

5.7 AfficheMatriceAPartirDeCoordonnees

Cette fonction est très utile. Elle permet d'afficher des matrices à partir d'ensemble de couples du type $m_{i,j} = val_{i,j}$.

Prenons un exemple pour mieux comprendre : l'ensemble

$$\{\{m_{1,1} = 0\}, \{m_{1,2} = 0, m_{2,2} = 0\}\}$$

affichera les matrices suivantes :

$$\begin{bmatrix} 0 & m_{1,2} \\ m_{2,1} & m_{2,2} \end{bmatrix}, \begin{bmatrix} m_{1,1} & 0 \\ m_{2,1} & 0 \end{bmatrix}.$$

Algorithm 9 AfficheMatriceAPartirDeCoordonnees

```
1  : AfficheMatriceAPartirDeCoordonnees := proc(n,Sol)
     ## condition pr que ca fonctionne :
     ## Sol = Ensemble de solutions composé d'ensemble de couples
m(i,j) = valij
4  :   local i,j,M,listeSol;
5  :   for i from 1 to nops(Sol) do
6  :     unassign(M);
7  :     for j from 1 to nops(Sol[i]) do
8  :       assign(subs(m=a,op(Sol[i][j])[1]),op(Sol[i][j])[2]);
9  :     od;
10 :     M := matrix(n,n,a);
11 :     print(eval(M));
12 :   od;
13 :   end :
```

Explication du code

1 : Déclaration de la procédure. Nous prenons comme paramètres n, la dimension des matrices carrées à afficher et *Sol*, l'ensemble des matrices comme décrit ci-dessus.

4 : Nous déclarons les variables locales nécessaires.

5 : Nous allons prendre chaque élément de l'ensemble. Chaque élément va correspondre à un ensemble de couples.

6 : Si la variable M existe on la libére.

7 : Pour chaque couple de l'ensemble courant :

8 : C'est un peu compliqué. Décomposons par le centre :

- `op(Sol[i][j])` nous donne les éléments composant le couple en cours (*exemple* :{$m_{1,1}$,0} pour `Sol[i][j] = (`$m_{1,1}$`=0)`) ;
- `op(Sol[i][j])[1]` représente le côté gauche du couple, donc la variable $m_{1,1}$;
- `op(Sol[i][j])[2]` représente le côté droit, donc la valeur à attribuer ;
- `subs(m=a,op(Sol[i][j])[1])` permet de remplacer par exemple $m_{1,1}$ par $a_{1,1}$;
- `assign(subs(m=a,op(Sol[i][j])[1]),op(Sol[i][j])[2])` assigne le deuxième membre dans la variable représentée par le premier (dans notre exemple : $a_{1,1} = 0$)

10 : On crée une matrice avec les éléments a. Vu que nous venons de les définir, M sera bien construite avec les valeurs données par les couples courants.

11 : On affiche la matrice M.

5.8 ReductionEnsembleMatrices

Lors de la courte exploration du cas de dimension 3, nous avons penser qu'il serait utile de réduire le nombre de matrices contenant un 1 dans chaque colonne et six variables quelconques. Si on ne le fait pas, nous nous retrouvons à étudier 27 matrices différentes !

Une idée simple pour réduire ce nombre est de ne garder que les matrices qui sont la sortie d'une transformation linéaire appliquée à une autre matrice. Effectivement, si nous nous trouvons face à une matrice non comprise dans le sous-ensemble qu'on aura construit, nous savons qu'une combinaison de transformations linéaires nous mènera à une matrice comprise dans ce sous-ensemble. De plus, il est intéressant de savoir quelle transfromation mène à une matrice éliminée.

Cet algorithme implémente toutes ces idées.

Algorithm 10 ReductionEnsembleMatrices

```
1  : ReductionEnsembleMatrices :=proc(MatrixA, Donnees)
2  :  local NonDoubles, NonPris, i, M, TableGamma, TableGammaB,
Mprime;
3  :  NonDoubles := {};
4  :  NonPris := {};
5  :  for i from 1 to nops(Donnees) do
6  :   M := Donnees[i];
7  :   TableGamma := Christoffel(M);
8  :   TableGammaB := TableGammaBarreLinear(MatrixA,TableGamma);
9  :   Mprime := TransformationGammaToMatrix(TableGammaB);
10 :   if not(true in map2(equal,op(Mprime),NonDoubles)) then
11 :    if not(true in map2(equal,op(M),NonDoubles)) then
12 :     NonDoubles := { op(NonDoubles), op(Mprime)};
13 :     NonPris := { op(NonPris), [op(M)."=>". op(Mprime)] };
14 :    fi;
15 :   fi;
16 :  od;
17 :  printf("Non Pris : ");
18 :  print(NonPris);
19 :  NonDoubles;
20 :  end :
```

Explication du code

1 : On déclare la procédure. Les paramètres sont la matrice *MatrixA* qui représente le changement de coordonnées linéaire et *Donnees* qui est l'ensemble de matrices à réduire.

2 : Déclaration des variables locales.

3 - 4 : On initialise deux ensembles vides : *NonDoubles* qui sera l'ensemble réduit et *NonPris* qui sera l'ensemble des matrices non

prises.

5 : On va parcourir l'ensemble des matrices à réduire.

6 : On attribue à M la matrice courante courante dans l'ensemble *Donnees*.

7 : On calcule les symboles de Christoffel à partir de la matrice M qu'on attribue à la variable *TableGamma*.

8 : On calcule ensuite ces mêmes symboles transformé par le changement de coordonnées linéaire défini par la matrice *MatrixA*. Les nouveaux symboles sont mis dans la variable *TableGammaB*.

9 : On récupère dans la variable *Mprime* la matrice qui provient des symboles de Christoffel compris dans *TableGammaB*.

10 - 11 : Si les deux matrices ne sont pas déjà comprises dans l'ensemble *NonDoubles* :

12 - 13 : On ajoute *Mprime* à *NonDoubles* et M à *NonPris* en indiquant vers quelle matrice M est transportée.

17 - 18 : On affiche l'ensemble *NonPris*.

19 : *NonDoubles* est renvoyé à l'utilisateur.

Chapitre 6

Conclusions

Nous terminerons cet ouvrage par résumer les buts atteints ainsi que toutes les idées acheminées tout au long de celui-ci. De plus, nous essaierons d'entrevoir les perspectives futures possibles pour la continuation de ce travail. En effet, par manque de temps ou pour raison de sujet trop éloigné du centre de ce mémoire, certaines parties ont été occultées. Nous pourrons voir notamment que certaines parties pourraient être automatisées.

6.1 Apports

Faisons le point sur ce qui a été fait et ce qu'a apporté ce mémoire.

Nous avons vu que la recherche sur les équations différentielles non linéaires n'est pas prête d'être terminée. En effet, nous en sommes qu'aux débuts prometteurs de celle-ci. Ce mémoire révèle une technique proposée par les chercheurs en mathématiques de l'Université Libre de Bruxelles. Nous avons exploré ensemble ce vaste domaine jusqu'aux limites que nous imposait le cadre de ce mémoire et évidemment le temps imparti.

Il en ressort une foule d'idées, concrétisées ou non, permettant de mieux analyser ces équations. Nous sommes partis du cas de dimension 2. Tout d'abord, celui-ci a été simplifié via des transformations linéaires en passant de quatre variables à deux variables.

Dès que la limite imposée par le linéaire fut atteinte, nous sommes passés aux transformations non linéaires. Celles-ci sont plus compliquées à rechercher. Nous avons donc mis au point une technique permettant de trouver des contraintes sur les fonctions composant le changement de coordonnées non linéaire ainsi que sur les éléments de la matrice 2 x 2 initiale. De ces contraintes ressortent des équations pour les 2 fonctions du changement de coordonnées.

Nous avons pu remarqué que notre méthode et les deux autres

méthodes présentées (via le tenseur de Riemann et la méthode théorique) donnent des résultats qui pourraient être complémentaires, c'est-à-dire que c'est peut-être une composition de ces 3 méthodes qui permettra de résoudre l'ensemble des équations différentielles non linéaires. Cela reste de la supposition mais mériterait une étude plus approfondie.

Nous trouvions intéressant de finir le cas de dimension 2 avant de commencer celui de dimension 3. Nous avons quand même essayer d'avancer dans cette matière. Nous avons pu ramener la matrice initiale de neuf variables à une autre de six variables via un changement de coordonnées linéaire déduit du cas de dimension 2. De plus, nous avons remarqué que nous pouvions étudier uniquement 7 matrices parmi les 27 possibles (composées de six variables quelconques et trois 1), ce qui permettra un gain de temps considérable.

Il est bon de également remarquer que la plupart des fonctions créées pour ce mémoire sont paramètrées par n, la dimension étudiée. Il n'y a donc nul besoin de réécrire les méthodes lorsqu'on désire étudier une nouvelle dimension. De plus, si une solution générale est trouvée, le cas de dimension n peut être étudié sans affecter à n un entier bien particulier, afin d'effectuer les calculs sous forme uniquement symbolique et donc vérifier la solution de manière tout à fait générale.

6.2 Développements futurs

Nous reprenons ici les objectifs futurs qui pourraient contribuer à construire une librairie plus complète avec des fonctions qui permettraient à la fois d'aller plus en profondeur dans l'étude des équations différentielles linéaires et à la fois d'automatiser le processus de recherche.

6.2.1 Questions en suspens

Nous n'avons pu, par faute de temps, finir le cas de dimension 3. Il serait évidemment intéressant de le terminer et ainsi peut-être, en le comparant avec le cas de dimension 2, trouver une forme générale pour la dimension n, qui est le but final.

Il serait intéressant d'établir le lien entre toutes les méthodes présentées (théorique, via le tenseur de Riemann et la nôtre) et d'étudier en profondeur les domaines qu'elles recouvrent chacune afin de se rendre compte de leur compatibilité effective.

Il faudrait également implémenter la méthode via le tenseur de Riemann car pour l'instant, seules les solutions sont identifiées, pas

le chemin pour y arriver !

6.2.2 Automatisation du processus

Dans le chapitre 4, nous nous sommes aidés de *Maple* afin de trouver les équations découlant des changements de coordonnées multiples. Mais nous avons nous-mêmes recherché dans celles-ci une manière de les simplifier. Pour le cas de dimension 2, cela ne pose pas trop de problème. Quand on arrive au cas de dimension 3, les choses se compliquent énormément. Il serait donc bon d'automatiser ce processus.

A notre avis, pour que ce processus donne des résultats corrects et dans des temps assez courts, il va falloir le travailler énormément.

L'idée de l'algorithme à fournir est le suivant :

Algorithm 11 Idée

```
Soit n la dimension à traiter
Soit M la matrice initiale de dimensions n x n.
Soient x_1, x_2,..., x_n les variables.
Soient f_1(x_1,x_2,...,x_n), f_2(x_1,x_2,...,x_n), ..., f_n(x_1,x_2,...,x_n) les
fonctions.

On calcule M' par un changement de coordonnées non linéaires
sur M. Ce changement de coordonnées est donné par le système
d'équations qu'est l'ensemble {f_1, f_2, ..., f_n}.
// On doit retirer les termes contenant des membres de la matrice
initiale

Pour MatriceI,MatriceJ de 1 jusque n

  Pour MI,MJ de 1 jusque n

    c := coefficiant du terme M[MI,MJ] dans M'[MatriceI,MatriceJ]

    Si c != 0 Alors

      On extrait les dérivées des fonctions qui jouent un rôle
dans ce coefficiant
      On met à jour la table des dérivées (voir plus loin)

    Fin Si
  Fin Pour
Fin Pour

Rechercher le nombre minimal de dérivées, contenues dans la table
construite, qui, si annulées, suppriment chaque membre de la
matrice initiale dans M'.
```

L'idée de la première partie de cet algorithme est de construire une table de la structure suivante :

$\partial_{x_1} f_1$	$\partial_{x_2} f_1$	...	$\partial_{x_1 x_2} f_1$	...	$\partial_{x_1 x_2 \ldots x_n} f_1$	$\partial_{x_1} f_2$	...
X		...	X	...		X	...
	X	...		...	X		...
	X	...		...	X		...
X		...		...	X	X	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Chaque ligne représente un membre de M'. Chaque "X" indique la présence de la dérivée (tête de colonne) dans le membre.

Le problème est de trouver le nombre minimal de dérivées (c'est-à-dire de colonnes) dont l'ensemble constituerait une colonne pleine de symboles "X".

Mais ce n'est malheureusement pas aussi simple. Il se peut que la solution que l'on trouve contienne des dérivées que l'on ne peut annuler car elles font parties d'un dénominateur qui s'annulerait à son tour. La solution n'est donc pas aisée et se doit d'être mûrement réfléchie.

De plus, cet algorithme n'est sûrement pas optimal. Il serait bon d'en recherche un meilleur.

6.3 Conclusion générale

Après un travail à la fois intéressant et prenant, j'ai remarqué que celui-ci reflètait exactement le travail d'un informaticien : être le traducteur entre les envies, les idées d'un groupe de personnes et l'ordinateur. C'est tellement enrichissant d'entrer dans un milieu que vous ne connaissez pas : vous ne pouvez qu'apprendre de nouvelles choses! Evidemment, ce n'est pas simple. Pour la réussite d'une telle immersion, il faut un dialogue constant entre les deux parties afin que les développeurs comprennent exactement ce que veulent les clients et que les clients voient exactement ce qu'il est possible de faire via l'informatique. Si une base solide est établie entre les deux parties, le projet a de grandes chances de se voir bien terminer un jour.

Pour ma part, j'ai eu énormément de chances de rencontrer les quatre professeurs qui ont contribués à ce mémoire. Ils ont été patients, intéressés et disponibles à chaque instant.

J'espère avoir été un maillon d'une chaîne dont la fin n'est pas prête d'être atteinte. Il y a apparemment tant à découvrir sur cette

matière que si mon travail peut faire avancer les choses, j'en serai extrêmement heureux.

Bibliographie

[1] Martin Braun, *Differential Equations and Their Applications (Third Edition) Short Version*, Ed Springer Verlag, ISBN 0-387-90847-1

[2] Léon Brenig, *On a constructive approach to the solution of dynamical systems*, IRMA Lectures in Mathematics and Theorectical Physics 3, 2003, Ed. F. Fauvet et C. Mitschi

[3] Léon Brenig, *Complete factorisation and analytic solutions of generalized Lokta-Voterra equations*, Phys. Lett. A133, 1988, 378.

[4] Léon Brenig, Alain Goriely, *Universal canonical forms for time-continuous dynamical systems*, Phys. Rev. A40, 1989, 4119.

[5] Alain Goriely, Léon Brenig, Algebric degeneracy and partial integrability for systems of ordinary differential equations, Phys. Lett. A145, 1990, 245.

[6] B. Hernández Bermejo, V. Fairén, Léon Brenig, Algebraic recasting of nonlinear systems of ODEs into universal formats, J. Phys. A31, 1998, 2415.

[7] M. Codutti, *NODES : A nonlinear ordinary differential equations solver, in : Proceedings of the ISSAC'92 Conference*, Berkeley 1992, ACM Press, Baltimore, MD, 1992, 69.

[8] André Heck, *Introduction to Maple*, Ed. Springer Verlag, ISBN 0-540-97662-0, ISBN 3-540-97662-0

[9] M.B. Monagan, K.O. Geddes, K.M. Heal, G. Labahn, S.M. Vorkoetter, *Maple V Programming Guide*, Ed. Springer, ISBN 0-387-94576-8

[10] Marco Codutti, *Non-linear Ordinary Differential Equations Solver. Algorithmes symboliques d'analyse d'intégrabilité pour les EDO*, Mémoire de licence, 1990, Bruxelles.

[11] Alain Goriely, *Transformations Quasi-Monomiales et Intégrabilité*, Mémoire de licence, 1989, Bruxelles

Annexe A

Documents supplémentaires

Les annexes sont composées de plusieurs documents :

1. L' entièreté du code produit avec ce mémoire.
   Pour une explication de celui-ci, se reporter au chapitre 5. Vous trouverez quelques différences entre la version présenté dans ce mémoire et le code mis ici en annexe. La version décrite en détail dans ce mémoire a été "allégée" de commades telles que certains *print* qui permettent juste d'afficher des résultats intermédiaires. Mais elles fournissent exactement les mêmes résultats.

2. Les résultats non repris dans le mémoire :

   (a) Les matrices de dimensions 3 x 3 que la méthode via le tenseur de Riemann peut résoudre.

   (b) La simplification des matrices 3 x 3 via des transformations linéaires.