Systèmes polynomiaux : que signifie « résoudre » ?

Université Lille 1
UE INFO 251

8 janvier 2007

Table des matières

Chapitre 1

Introduction

Équipe pédagogique

François	Boulier
Jean–Claude	Depannemaecker
Jean–Charles	Faugère
Michel	Petitot
François	Recher
Fabrice	Rouillier
Mohab	Safey El–Din
Alexandre	Sedoglavic
Éric	Wegrzynowski
Léopold	Weinberg

Ce cours est issu de l'exposé assuré par Fabrice Rouillier le 7 juillet 2002 pendant le « week–end d'initiation–formation au calcul formel » organisé à l'occasion de la conférence ISSAC 2002.

Il s'agit d'une introduction à la théorie et la pratique d'une chaîne logicielle développée par Fabrice Rouillier et Jean–Charles Faugère dans le cadre du projet INRIA[1] SPACES et destinée à la résolution réelle d'une certaine classe de systèmes d'équations polynomiales.

1.1 Résoudre

Résoudre un système d'équations linéaires à coefficients rationnels est facile en théorie : l'algorithme du pivot de Gauss permet de décider de l'existence et de l'unicité d'une solution. Il fournit la solution — nécessairement rationnelle — lorsqu'elle est unique. Il permet de décrire simplement l'ensemble des solutions lorsqu'il y en a une infinité.

Résoudre une équation polynomiale de degré deux et à coefficients rationnels est facile : on dispose de formules simples explicitant « par radicaux[2] » la ou les racines du polynôme.

[1] Institut National de Recherche en Informatique et en Automatique.

[2] On dit qu'une racine d'un polynôme en une indéterminée est exprimable par radicaux si elle peut se noter à partir

On a chacun eu l'occasion, lors de sa formation, d'étudier les types d'équations ci–dessus ainsi que les méthodes correspondantes et on pourrait être tenté de croire que cette situation idyllique se généralise : que quelle que soit la sorte d'équation, il existe une méthode de résolution simple et satisfaisante.

Eh bien non.

On a appris au vingtième siècle que dans beaucoup de « domaines naturels » les systèmes d'équations étaient insolubles. Insolubles dans le sens où il est démontré qu'il ne peut exister aucun algorithme qui prenne en entrée un système d'équations quelconque du domaine et se contente même de décider[3] si le système admet au moins une solution.

C'est ce que montrent en particulier le théorème de Matijasevic et celui de Caviness, Richardson et Matijasevic (cités plus bas).

Ensuite même dans les domaines où on dispose d'algorithmes de résolution (le domaine des systèmes d'équations polynomiales à coefficients rationnels en est un), il reste l'épineuse question : « comment présenter les solutions (au lecteur, à l'utilisateur de logiciel etc...) » ?

Le théorème d'Abel illustre cette difficulté.

Ces résultats limitatifs prouvent bien l'intérêt de l'algorithme de Buchberger pour la simplification de systèmes d'équations polynomiales que nous étudierons en deuxième partie du cours. Ils montrent aussi que la méthode de résolution réelle de polynômes en une indéterminée que nous étudierons en première partie fournit un résultat quasiment optimal.

Théorème 1 *(Matijasevic, 1970)*

Le dixième problème posé par Hilbert en 1900 n'admet pas de solution.

En d'autres termes, il ne peut exister aucun algorithme qui prenne en entrée une équation polynomiale en un nombre n quelconque d'indéterminées, à coefficients rationnels et qui décide si cette équation admet une solution rationnelle (c'est–à–dire une solution formée de n nombres rationnels).

Le théorème de Matijasevic a pour conséquence (entre autres !) le théorème suivant.

Théorème 2 *(Caviness, Richardson et Matijasevic, 1968, 1970)*

Soit $\mathscr{F}$ l'ensemble de toutes les formules qu'on peut construire avec des rationnels, π, un symbole x, les opérations $+$, $\times$, sin et abs. Par exemple

$$\sin(2\pi + 3), \quad |x| + 1/2, \quad \ldots$$

Il ne peut exister aucun algorithme qui prenne en entrée deux formules quelconques de $\mathscr{F}$ et décide si ces formules représentent la même fonction de $\mathbb{R}$ dans $\mathbb{R}$.

Théorème 3 *(Abel, 1824)*

A partir du degré 5, la plupart des équations polynomiales en une indéterminée et à coefficients rationnels ne peuvent se résoudre par radicaux.

---

des coefficients du polynôme, au moyen des opérations $+$, $\times$ et extraction de racine enième ($n = 2$ dans le cas des polynômes du second degré).

[3] C'est–à–dire réponde à coup sûr par oui ou non.

FIG. 1.1 – Niels Henrik Abel, 1902–1829.

L'équation $x^5 - x - 1 = 0$ est de celles–là.

Il est intéressant de remarquer la façon dont les logiciels de calcul formel contournent cette difficulté. À la question : quelles sont les racines du polynôme $x^5 - x - 1$? le logiciel MAPLE répond : « ce sont tous les x tels que $x^5 - x - 1 = 0$ ».

```
solve (x^5 - x - 1);
                                  5
                        RootOf(_Z  - _Z - 1)
```

1.2 La chaîne GB + RS

Cette chaîne logicielle[4] prend en entrée un système d'équations polynomiales en plusieurs indéterminées et à coefficients rationnels. Elle produit en sortie la liste de toutes les solutions réelles du système, sous réserve que l'ensemble des solutions complexes soit fini. La chaîne détermine automatiquement si les systèmes en entrée ont aucune, un nombre fini ou une infinité de solutions.

Les méthodes mises en œuvre sont purement algébriques. Tous les calculs sont exacts. Plus précisément, les algorithmes mis en œuvre ne procèdent jamais à aucune approximation. Ils ne « perdent » ou n'« inventent » aucune solution en cours de calcul. Les coefficients des expressions manipulées sont gardés sous la forme de nombres rationnels : ils ne sont jamais arrondis sauf éventuellement à la fin des calculs, pour rendre les résultats plus lisibles.

La chaîne est composée de deux grandes parties : la première consiste à simplifier le système en entrée, la seconde à résoudre le système simplifié. La première est le fait du logiciel GB, dû à Jean–Charles Faugère, qui implante un algorithme de calcul de « bases de Gröbner », la seconde est le fait du logiciel RS, dû à Fabrice Rouillier. Ces deux logiciels ont été écrits en langage C.

[4]Par chaîne logicielle, on entend une suite de logiciels à exécuter les uns à la suite des autres, chacun prenant en entrée le résultat calculé par celui qui le précède.

Il est remarquable que la théorie mise en œuvre relevait du domaine de la recherche au début des années 90. Elle était alors considérée comme inapplicable. Ce sont essentiellement des progrès informatiques qui ont fourni une chaîne logicielle capable de résoudre des problèmes importants, inaccessibles à toute autre méthode : GB + RS a permis a l'INRIA de décrocher un contrat industriel avec une PME des Vosges, spécialisée en machines–outils, sur un problème de commande de robots parallèles.

1.3 Le cours

Il s'articule en deux grandes parties lui aussi. Il commence par le chapitre 2 et l'étude d'une méthode de résolution réelle de polynômes en une indéterminée et à coefficients rationnels (avec une section dédiée à l'arithmétique par intervalles). On y étudie la méthode implantée dans RS. Il se poursuit, avec le chapitre 3, par l'étude de la théorie des bases de Gröbner qui sous–tend GB.

Le choix de présentation peut sembler surprenant : on commence par la fin et on termine par le début de la chaîne mais nous avons cru bon d'aborder l'étude par les polynômes en une indéterminée qui sont plus familiers aux étudiants de deuxième année de licence et d'introduire les notions propres aux polynômes en plusieurs indéterminées plus progressivement, par le biais notamment d'un mini–projet de tracé de courbes algébriques en deux indéterminées.

Nous avons choisi également de présenter GB + RS comme une chaîne logicielle « presse bouton » c'est–à–dire une chaîne totalement automatisée de résolution de systèmes d'équations algébriques. Ce choix permet de simplifier au maximum la théorie exposée dans le cours, qui est déjà considérable pour une licence mais il est un peu trompeur : la vraie chaîne GB + RS ressemble davantage à une collection de logiciels implantant des algorithmes variés. C'est l'utilisateur qui est chargé de déterminer l'algorithme le mieux adapté à son problème.

Deux chapitres complètent ce cours.

Le chapitre 4 présente un algorithme pour le pgcd de deux polynômes en une indéterminée et à coefficients rationnels. Le calcul du pgcd de deux polynômes est une fonctionnalité nécessaire pour la partie RS. Les méthodes employées dans l'algorithme présenté s'appuient sur des techniques de calcul modulaire, qu'on retrouve, en particulier, en cryptographie. Le calcul modulaire constitue une introduction simple à la théorie des idéaux et aux bases de Gröbner.

Le chapitre 5 présente, à titre de comparaison, la méthode de Newton pour la résolution numérique des équations. Elle constitue, de loin, la méthode la plus utilisée pour résoudre les systèmes d'équations. On la présente dans le cas d'une équation en une inconnue et dans le cas de deux équations à deux inconnues.

le support de cours est émaillé de portraits. Certaines photographies ont été prises sur des pages web personnelles. La plupart des portraits « historiques » ont été prises sur le site web des biographies de mathématiciens de l'université de Saint–Andrews [17].

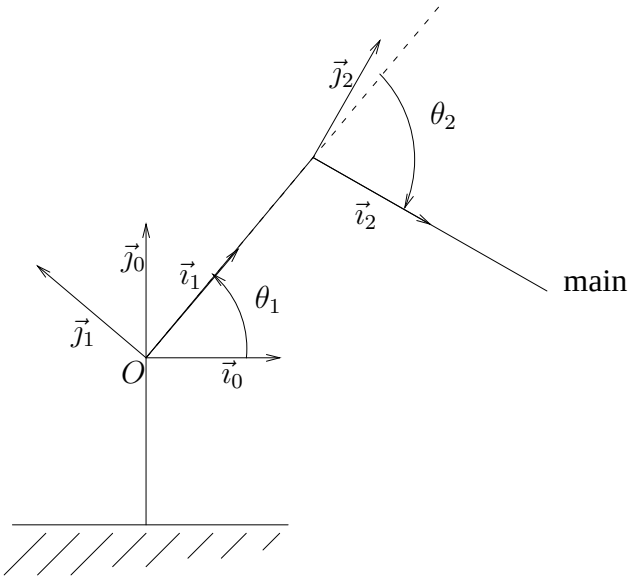
1.4 Commande de robots

On conclut cette introduction en présentant le problème de la commande de deux types de robots (un robot série planaire et un robot parallèle) qui illustrent l'utilité du logiciel. Cette partie de l'introduction s'appuie sur les notions développées dans les chapitres suivants. Il est donc normal de ne pas bien la comprendre à la première lecture.

1.4.1 Un robot série planaire

Cet exemple est extrait du livre [6].

Il s'agit d'un bras formé de deux segments rigides reliés à une base fixe. La base est symbolisée par un segment de droite vertical. Une main est située à l'extrémité du dernier segment. Deux joints circulaires articulent le premier segment avec la base et les deux segments entre eux. Le bras se déplace dans un plan.



On commande les deux angles θ_1 et θ_2. On souhaite contrôler la position de la main.

La modélisation est la suivante. On attache un repère global $(O,\, \vec{\imath}_0,\, \vec{\jmath}_0)$ à la base ainsi qu'un repère local $(O_k,\, \vec{\imath}_k,\, \vec{\jmath}_k)$ à chacun des deux joints ($k = 1,\, 2$). Les repères locaux se déplacent avec les segments. Le vecteur $\vec{\imath}_k$ est placé dans l'axe défini par le k–ème segment ; le vecteur $\vec{\jmath}_k$ est orthogonal à $\vec{\imath}_k$; on définit θ_k comme l'angle orienté qui envoie $\vec{\imath}_{k-1}$ sur $\vec{\imath}_k$ (pour $k = 1,\, 2$).

On distingue le problème géométrique direct du problème géométrique inverse.

Le problème géométrique direct consiste à déterminer les coordonnées $(x_0,\, y_0)$ de la main dans le repère global connaissant θ_1 et θ_2.

Le problème géométrique inverse consiste à déterminer les angles θ_1 et θ_2 connaissant les coordonnées $(x_0,\, y_0)$ de la main dans le repère global.

Le problème inverse est plus difficile que le problème direct. Il admet zéro (quand la main est « trop loin »), une (quand le bras est « tendu ») ou deux solutions (qui se déduisent l'une de l'autre par symétrie).

Mise en équation du problème (on nomme ℓ_1 et ℓ_2 les longueurs des segments).

$$\begin{cases} \ell_2(\cos\theta_1\cos\theta_2 - \sin\theta_1\sin\theta_2) + \ell_1\cos\theta_1 - x_0 = 0, \\ \ell_2(\cos\theta_1\sin\theta_2 + \cos\theta_2\sin\theta_1) + \ell_1\sin\theta_1 - y_0 = 0. \end{cases}$$

Les équations ne sont pas polynomiales. Pour les rendre polynomiales, on introduit quatre nouvelles indéterminées c_k et s_k pour désigner le cosinus et le sinus de l'angle θ_k qu'on lie par les relations bien connues : $c_k^2 + s_k^2 = 1$ (pour $k = 1, 2$) :

$$\mathscr{S}\begin{cases} \ell_2(c_1c_2 - s_1s_2) + \ell_1c_1 = x_0, \\ \ell_2(c_1s_2 + c_2s_1) + \ell_1s_1 = y_0, \\ c_1^2 + s_1^2 = 1, \\ c_2^2 + s_2^2 = 1. \end{cases}$$

Ce système est suffisamment simple pour pouvoir être résolu avec MAPLE. On pose que les deux segments sont de longueur 1.

On place pour commencer la main au point de coordonnées $(1/3,\ 1/2)$. L'algorithme de calcul de bases de Gröbner de MAPLE transforme le système en un système équivalent (ayant mêmes solutions) sous « forme résolue » : le système simplifié comporte une équation de degré deux en l'indéterminée s_1 uniquement ; les autres équations expriment les autres indéterminées comme des fonctions polynômes des solutions de la première équation. Comme les fonctions polynômes ont des coefficients rationnels (c'est une propriété de l'algorithme de bases de Gröbner : si on lui donne un système à coefficients rationnels, il produit un système à coefficients rationnels), toute solution réelle de la première équation fournit une racine réelle du système. Le discriminant de l'équation de degré deux est positif : le système a donc deux solutions réelles.

```
with (Groebner):
p1 := l2*(c1*c2 - s1*s2) + l1*c1 - x0:
p2 := l2*(c1*s2 + s1*c2) + l1*s1 - y0:
p3 := c1^2 + s1^2 - 1:
p4 := c2^2 + s2^2 - 1:
S := [p1,p2,p3,p4]:
l1 := 1: l2 := 1: x0 := 1/3: y0 := 1/2:
G1 := gbasis (S, plex (c2,s2,c1,s1));

                          2
          G1 := [1872 s1  - 407 - 936 s1, 24 c1 + 36 s1 - 13,

                 48 s2 + 52 s1 - 13, 72 c2 + 59]

discrim (G1 [1], s1);
                                   3923712
```

On place maintenant la main sur le cercle de rayon $\ell_1 + \ell_2$ (qui est égal à deux). Le bras est donc tendu. On obtient à nouveau un système sous « forme résolue ». Le polynôme de degré deux en s_1 a une racine double.

```
x0 := 2: y0 := 0:
G2 := gbasis (S, plex (c2,s2,c1,s1));
                              2
                  G2 := [s1 , c1 - 1, 2 s1 + s2, c2 - 1]
```

On place maintenant la main « trop loin ». On obtient encore un système sous « forme résolue ». Le discriminant du polynôme de degré deux en s_1 est négatif : le système a deux solutions complexes, aucune solution réelle.

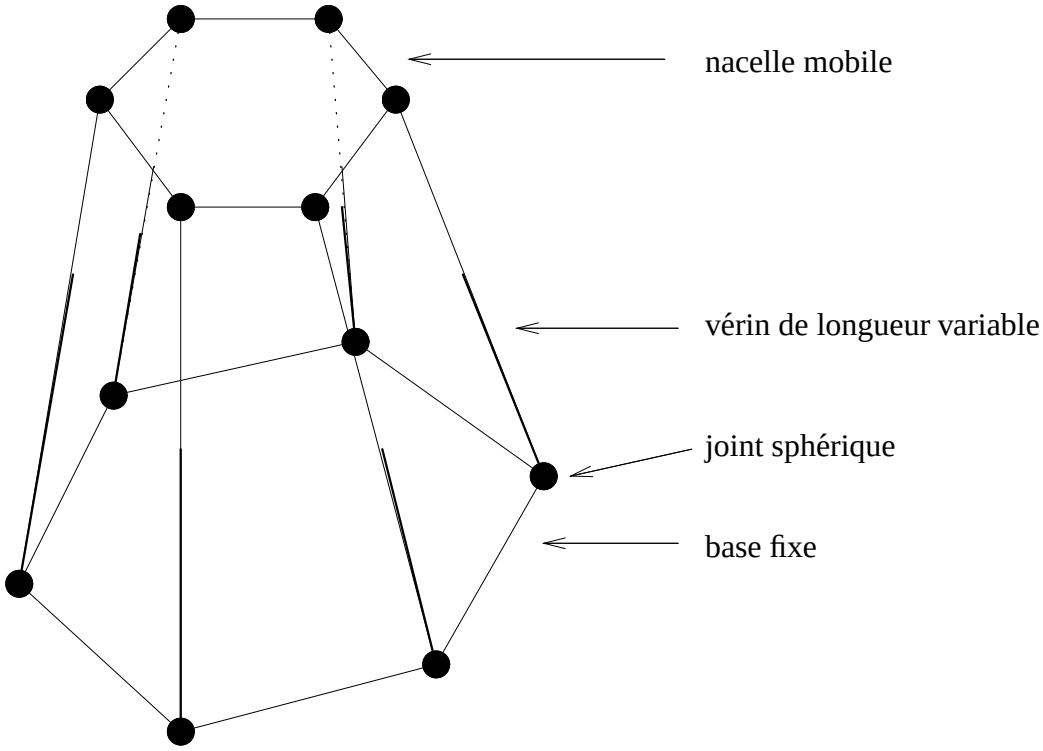
```
x0 := 3: y0 := 1:
G3 := gbasis (S, plex (c2,s2,c1,s1));
                    2
       G3 := [5 s1  + 8 - 5 s1, 3 c1 + s1 - 5, 3 s2 + 10 s1 - 5, c2 - 4]

discrim (G3 [1], s1);
                                    -135
```

1.4.2 Un robot parallèle

C'est sur un problème du type de celui–ci que la chaîne GB + RS a permis à l'INRIA de décrocher un contrat avec une PME spécialisée en machines–outils.

Un robot parallèle (le modèle le plus général) est constitué d'une nacelle hexagonale reliée par six vérins à une base fixe. Les vérins sont attachés à la base et à la nacelle par des joints sphériques. On contrôle la position de la nacelle en contrôlant la longueur des vérins.



Les robots parallèles présentent l'avantage d'être rapides, précis et de permettre la manipulation d'outillage lourd. Ils commencent à être utilisés en robotique pour l'usinage à grande vitesse. On les trouve aussi dans les simulateurs de vol, les mécanismes d'orientation d'antennes et même de suspension de voitures.

On distingue ici aussi le problème géométrique direct du problème géométrique inverse.

Le problème géométrique direct consiste à déterminer la position de la nacelle (dans un repère global, fixe, lié à la base) connaissant la longueur des six vérins.

Le problème géométrique inverse consiste à déterminer la longueur des six vérins connaissant la position de la nacelle (dans le repère global).

Dans le contexte des robots parallèles, le problème inverse est facile : il n'y a au plus qu'une seule solution. Le problème direct est difficile : en général, pour une longueur donnée des six vérins, la nacelle peut prendre 40 positions différentes.

Les deux problèmes sont importants.

Lorsqu'on souhaite commander le robot, le problème qui se pose est le suivant : connaissant la trajectoire qu'on souhaite faire suivre à la nacelle, déterminer la commande à appliquer au robot. On cherche une commande sous la forme d'un jeu de six fonctions du temps $\ell_i(t)$; la fonction $\ell_i(t)$ donne la longueur du vérin i à l'instant t.

Pour résoudre le problème, la solution usuelle consiste à discrétiser[5] la trajectoire désirée sous la forme d'une suite finie de points $p_1, \ldots, p_n$ où chaque p_i est (par exemple) un triplet $(x_i,\, y_i,\, z_i)$ de coordonnées dans le repère global. Pour chaque point p_i, on résout le problème géométrique inverse, ce qui donne un jeu de longueurs $(\ell_{i,1}, \ldots, \ell_{i,6})$. En reliant « comme on peut » la suite de longueurs $\ell_{1,k}, \ldots, \ell_{n,k}$ à imposer au k–ème vérin, on obtient une fonction $\ell_k(t)$.

Malheureusement, si on applique la commande trouvée à un robot parallèle, on s'aperçoit que le robot ne suit pas nécessairement la trajectoire voulue. On s'aperçoit qu'on peut même casser le robot !

Il y a plusieurs raisons à cela.

1. Pour certains jeux de longueurs des vérins, la nacelle peut prendre des positions très proches. Si la commande appliquée aux vérins passe par de tels jeux de longueurs, la nacelle peut quitter la trajectoire qu'on souhaite lui faire suivre et bifurquer sur une autre.
2. Une toute petite modification des longueurs des jambes peut faire parcourir une très grande distance à la nacelle.
3. Rien ne garantit que la trajectoire qu'on souhaite imposer à la nacelle est continue. Il se peut fort bien que deux positions successives p_i et p_{i+1} de la nacelle se trouvent sur deux trajectoires admissibles différentes.

Pour éviter ces écueils, il faut (au minimum mais ce n'est pas forcément suffisant) résoudre le problème géométrique direct pour chaque jeu de longueurs $(\ell_{i,1}, \ldots, \ell_{i,6})$.

On peut modéliser le robot par un système de neuf équations polynomiales en neuf inconnues.

Pour les deux premiers écueils, il faut être capable de trouver toutes les solutions réelles du système même dans le cas (surtout dans le cas) où plusieurs solutions réelles sont très proches l'une de l'autre.

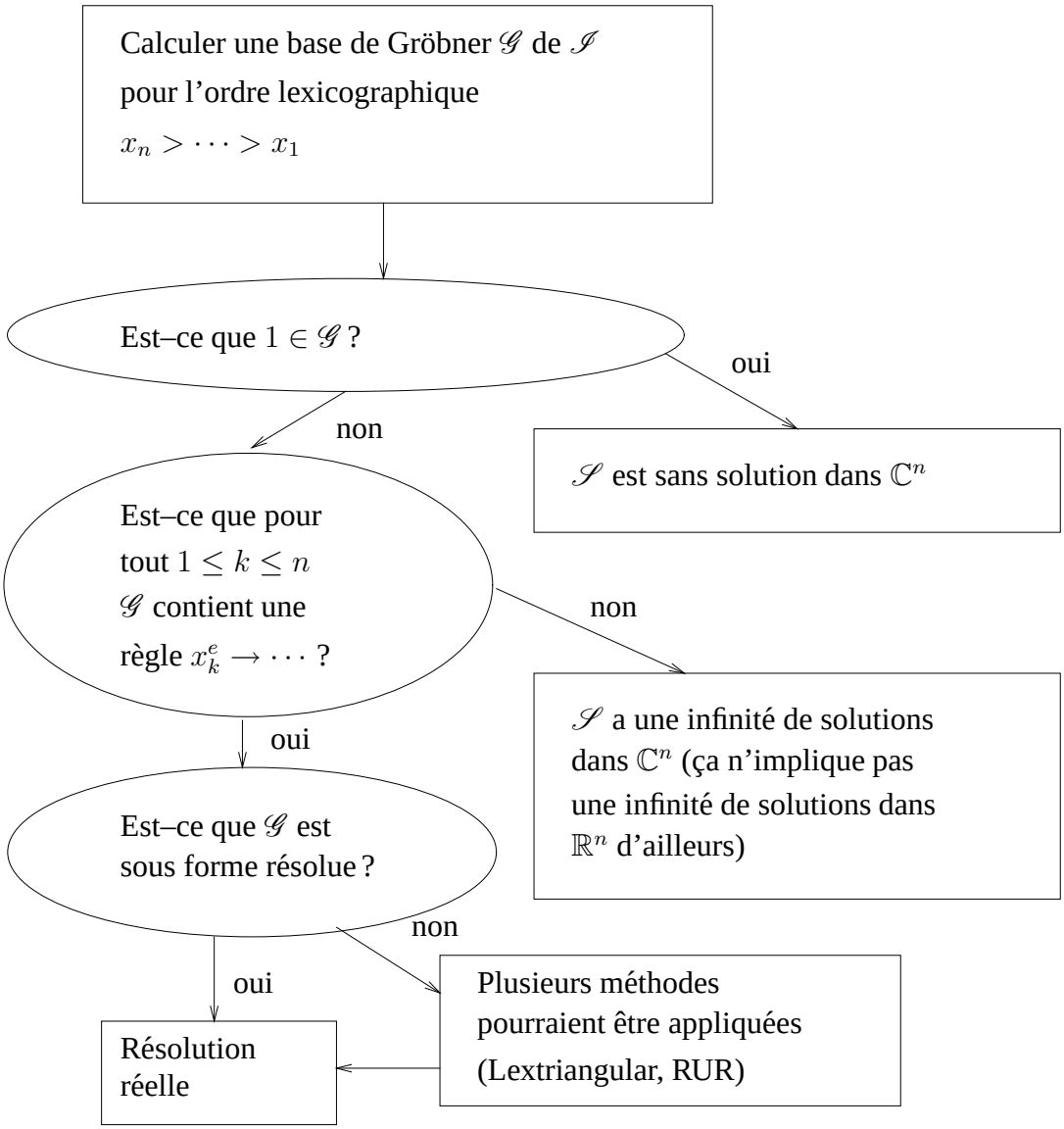
[5] discrétiser une courbe signifie l'approximer par une suite finie de points.

Pour le troisième, il faut être capable de décider si une solution complexe « presque réelle » (c'est–à–dire de la forme $a + i\,b$ avec b très petit en valeur absolue) est complexe ou pas (la discontinuité d'une trajectoire se traduit par le fait que deux solutions réelles du système se rapprochent l'une de l'autre, deviennent une position réelle double puis deux solutions complexes).

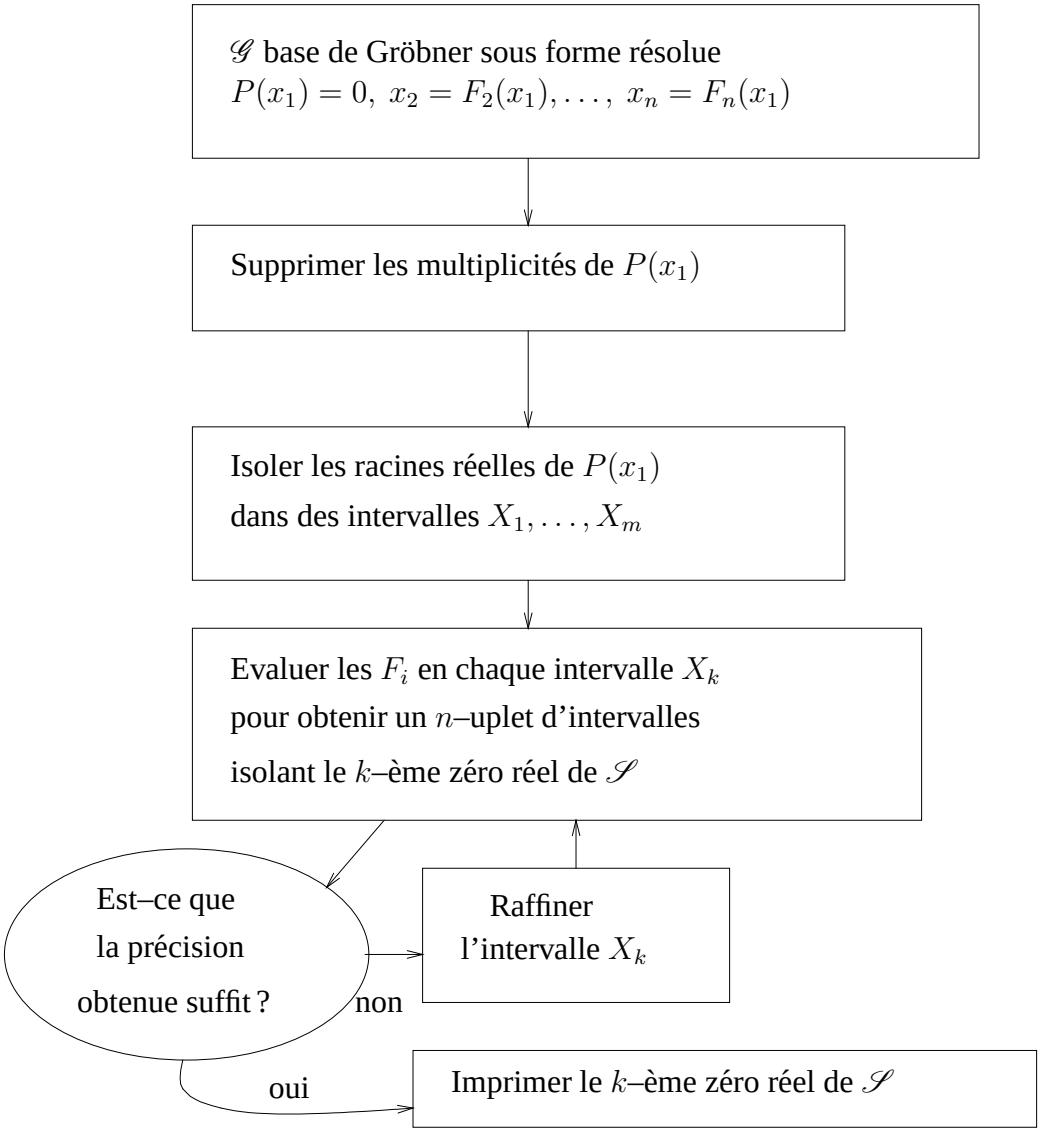
Le système à résoudre est trop difficile pour MAPLE. Une fois mis sous « forme résolue », il comporte une équation en une indéterminée de degré 40 (puisqu'il y a 40 solutions). Les coefficients des polynômes qui constituent la base de Gröbner font plus d'une centaine de chiffres chacun.

1.5 Schéma général de la partie GB

Soient $\mathscr{S}$ un système de polynômes de l'anneau $\mathbb{A} = \mathbb{Q}[x_1, \ldots, x_n]$ et $\mathscr{I}$ l'idéal qu'il engendre.



1.6 Schéma général de la partie RS



Chapitre 2

Résolution réelle

Dans cette partie, on s'intéresse au problème du calcul des racines réelles d'un polynôme $P \in \mathbb{Q}[x]$ en une indéterminée et à coefficients rationnels. Le degré de P peut être élevé. Ses coefficients peuvent être gros. On verra dans la deuxième partie comment de tels polynômes apparaissent naturellement lors de la résolution de systèmes d'équations en plusieurs indéterminées.

Le théorème d'Abel montre qu'on ne peut pas espérer résoudre P par radicaux. Même dans les cas où c'est possible, l'utilisation de radicaux serait à proscrire d'ailleurs : décider si deux telles expressions désignent un même nombre n'est pas facile ; une expression comprenant des radicaux ne s'évalue pas facilement numériquement.

L'algorithme que nous allons étudier est un algorithme d'isolation des racines réelles : il produit en sortie une liste d'intervalles (autant d'intervalles qu'il y a de racines, exactement une racine par intervalle) dont les bornes sont des nombres exacts (des rationnels), raffinables à volonté.

2.1 Suppression des multiplicités

2.1.1 Rappels mathématiques

Définition 1 *(anneau)*

Un ensemble $\mathbb{A}$ muni d'une addition $+$ et d'une multiplication $\times$ possède une structure d'anneau *pour ces opérations si*

- *$\mathbb{A}$ possède une structure de groupe commutatif pour l'addition*
- *la multiplication est associative*
- *la multiplication est distributive à gauche et à droite par rapport à l'addition.*

Dans ce texte, on ne considère que des anneaux commutatifs et unitaires, c'est–à–dire des anneaux dont la multiplication est commutative et qui contiennent un élément neutre (noté 1) pour cette opération. On dit abusivement qu'un ensemble $\mathbb{A}$ est un anneau lorsqu'il n'y a aucune ambiguité sur les opérations d'addition et de multiplication. Les ensembles $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Q}[x]$ sont des *anneaux*. Par contre, l'ensemble $\mathbb{N}$ des entiers naturels n'est pas un anneau.

Définition 2 *(corps)*

Un élément A d'un anneau $\mathbb{A}$ est dit inversible *s'il existe un élément $\bar{A} \in \mathbb{A}$, appelé* inverse *de A, tel que $A\,\bar{A} = 1$. Un anneau dont tous les éléments non nuls sont inversibles est un* corps.

Les anneaux $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ sont des corps. Les anneaux $\mathbb{Z}$, $\mathbb{Q}[x]$ n'en sont pas.

Définition 3 *(divisibilité)*
Soient $\mathbb{A}$ un anneau et $A,\ B$ deux de ses éléments. On dit que A divise *B (ce qu'on note $A \mid B$) s'il existe $C \in \mathbb{A}$ tel que $B = A\,C$.*

On s'intéresse aux anneaux de polynômes $\mathbb{K}[x]$ en une indéterminée x et à coefficients dans le corps $\mathbb{K}$ où $\mathbb{K} = \mathbb{Q}$, $\mathbb{R}$ ou $\mathbb{C}$.

2.1.2 Division euclidienne, pgcd dans $\mathbb{K}[x]$

FIG. 2.1 – Euclide 325–265 avant J.-C. (environ)

Proposition 1 *(division euclidienne)*
Soient F et $G \neq 0$ deux polynômes de $\mathbb{K}[x]$. Il existe un unique couple de polynômes $(Q,\ R)$ tels que $F = G\,Q + R$ et $\deg R < \deg G$. Les polynômes Q et R sont appelés le quotient *et le* reste *de la division euclidienne de F par G.*

On déduit de la proposition précédente que $G \mid F$ si et seulement si le reste de la division euclidienne de F par G est nul. Les fonctions MAPLE *quo* et *rem* implantent le calcul du quotient et du reste de la division euclidienne de deux polynômes en une indéterminée.

Définition 4 *(pgcd)*
Soient F et G deux polynômes de $\mathbb{K}[x]$. On appelle plus grand commun diviseur *(ou* pgcd*) de F et de G « le » diviseur commun de F et de G de degré le plus élevé. On le note $P = F \wedge G$.*

Il y a d'autres façons de définir le pgcd de deux polynômes. Voir par exemple la définition 26, page 63. Le pgcd de F et de G est défini au produit près par un élément inversible de $\mathbb{K}[x]$, c'est–à–dire un élément non nul de $\mathbb{K}$. Il n'est donc pas unique. Par abus de langage on parle tout de même du pgcd de F et de G. Il peut se calculer au moyen de l'algorithme d'Euclide, qui repose sur la proposition suivante.

Proposition 2 *(algorithme d'Euclide)*

Si $G = 0$, le plus grand commun diviseur de F et de G est F.

Si $G \neq 0$, l'ensemble des diviseurs communs de F et de G est égal à l'ensemble des diviseurs communs de G et de R où R désigne le reste de la division euclidienne de F par G.

```
function Euclide (F, G)
begin
  R := F
  while R ≠ 0 do
    R := le reste de la division euclidienne de F par G
  od
  R
end
```

En même temps que le pgcd P de F et de G, il est possible de calculer une *identité de Bézout* (le texte historique est [3]) entre F et de G, c'est–à–dire deux polynômes U et V tels que

$$F\,U + G\,V = P.$$

La fonction suivante effectue un calcul en parallèle sur les trois composantes de deux vecteurs $\mathscr{U} = (U_1, U_2, U_3)$ et $\mathscr{V} = (V_1, V_2, V_3)$. Le calcul effectué sur la troisième composante correspond à l'algorithme d'Euclide. Voici trois invariants de la boucle ci–dessous.

1. $U_1\,F + U_2\,G = U_3$
2. $V_1\,F + V_2\,G = V_3$,
3. l'ensemble des diviseurs communs de F et de G est égal à l'ensemble des diviseurs communs de U_3 et de V_3.

```
function Euclide_étendu (F, G)
begin
  𝒰 := (1, 0, F)
  𝒱 := (0, 1, G)
  while V₃ ≠ 0 do
    Q := le quotient de la division euclidienne de U₃ par V₃
    𝒯 := 𝒱
    𝒱 := 𝒰 − Q 𝒱
    𝒰 := 𝒯
  od
  𝒰
end
```

FIG. 2.2 – Etienne Bézout, 1730–1783.

En MAPLE, l'algorithme d'Euclide étendu pour les polynômes en une indéterminée est implanté par la fonction *gcdex*. La fonction *gcd* implante un algorithme plus général : le pgcd de deux polynômes en plusieurs indéterminées, pour lequel il n'existe pas de version « étendue ».

```
F := x^2-2:
G := x:
P := gcdex (F, G, x, 'U', 'V');
                                        P := 1

U, V;
                                      -1/2, x/2

U*F + V*G;
                                           1
```

En pratique, pour calculer le pgcd de deux polynômes $F, G \in \mathbb{Q}[x]$, les logiciels de calcul formel emploient bien l'algorithme d'Euclide mais d'une façon plus subtile que ce qui est décrit ci–dessus, pour éviter la croissance des coefficients des restes intermédiaires. Ce sujet est développé dans le chapitre 4.

Et qu'en est–il des polynômes à coefficients dans $\mathbb{R}$ ou dans $\mathbb{C}$? D'un point de vue algorithmique, ces corps n'existent pas. En effet, la plupart des réels (ou des complexes) sont impossibles à représenter et donc à manipuler en machine. Raisonnons par l'absurde et supposons qu'on dispose d'une structure de données permettant de représenter les nombres réels en machine. On pourrait alors numéroter les nombres réels : il suffirait de numéroter les structures de données (en les ordonnant d'abord par taille croissante, puis en ordonnant les structures de données d'une taille donnée suivant un principe quelconque). Cette numérotation fournirait une bijection de $\mathbb{N}$ sur $\mathbb{R}$, dont Cantor a prouvé à la fin du dix–neuvième siècle qu'elle ne pouvait pas exister. On dit que $\mathbb{R}$ n'est pas « dénombrable ». Des arguments similaires montrent que les nombres réels sont incalculables pour la plupart.

2.1.3 Irréductibilité, multiplicité

Définition 5 *Un polynôme $P \in \mathbb{K}[x]$ est* irréductible *s'il n'appartient pas à $\mathbb{K}$ et s'il n'est divisible par aucun polynôme $F \in \mathbb{K}[x]$ tel que $0 < \deg F < \deg P$.*

La notion d'irréductibilité est subtile parce qu'elle dépend fortement de l'anneau de polynômes considéré : le polynôme $x^2 - 2$ est irréductible dans $\mathbb{Q}[x]$ mais pas dans $\mathbb{R}[x]$.

Les polynômes de degré 1 sont irréductibles dans $\mathbb{K}[x]$ pour tout corps $\mathbb{K}$. Dans $\mathbb{C}[x]$, ce sont les seuls. Dans $\mathbb{R}[x]$, il existe d'autres polynômes irréductibles, qui sont de degré 2 ($x^2 + 1$ par exemple). Dans $\mathbb{Q}[x]$ il existe des polynômes irréductibles de tous les degrés.

Théorème 4 *Tout polynôme $P \in \mathbb{K}[x]$ admet une factorisation en un produit de puissances de polynômes irréductibles de $\mathbb{K}[x]$*

$$P = F_1^{d_1} \cdots F_r^{d_r}.$$

Cette factorisation est unique au produit des F_i par un élément inversible de $\mathbb{K}[x]$ et à l'ordre des facteurs près. On l'appelle la factorisation complète *de P dans $\mathbb{K}[x]$.*

Les polynômes F_i sont appelés les *facteurs irréductibles* de P. L'exposant d_i est la *multiplicité du facteur F_i*. Si d_i vaut 1, on dit que F_i est un facteur irréductible *simple* de P. Ce qui précède implique que tout polynôme P admet dans $\mathbb{C}[x]$ une unique factorisation

$$P = (x - a_1)^{d_1} \cdots (x - a_r)^{d_r}.$$

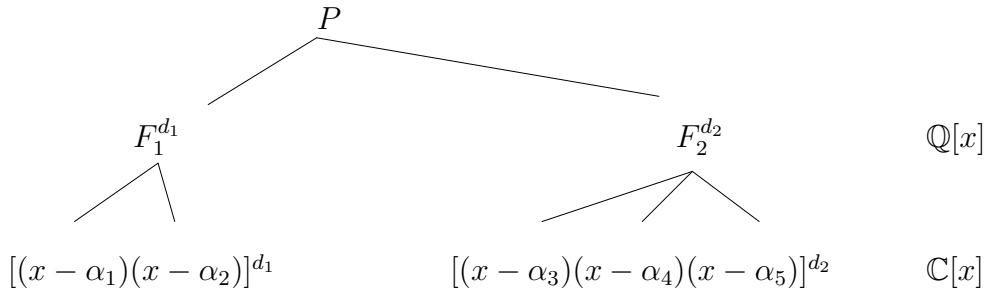
C'est le théorème de d'Alembert. Les nombres complexes a_i sont les *racines* de P. L'exposant d_i est la *multiplicité de la racine a_i*. Si d_i vaut 1, on dit que a_i est une racine *simple* de P.

Il est évident que a est une racine de P de multiplicité d si et seulement si $(x-a)$ est, dans $\mathbb{C}[x]$, un facteur irréductible de P de multiplicité d. Il y a aussi correspondance entre les multiplicités des facteurs irréductibles de P dans $\mathbb{K}[x]$ et celles des racines de P. C'est ce que montrent les propositions suivantes, dont les preuves sont laissées en exercice.

Proposition 3 *Un polynôme irréductible P de $\mathbb{K}[x]$ n'a que des racines simples.*

Proposition 4 *Deux polynômes irréductibles distincts de $\mathbb{K}[x]$ n'ont pas de racine commune.*

Considérons par exemple le cas d'un polynôme P de $\mathbb{Q}[x]$ admettant une factorisation complète dans $\mathbb{Q}[x]$ de la forme $P = F_1^{d_1} F_2^{d_2}$. Mettons que F_1 ait trois racines et que F_2 en ait deux. On voit que les racines de F_1 sont des racines de P de multiplicité d_1 et que celles de F_2 sont des racines de P de multiplicité d_2. On dit que la factorisation complète de P dans $\mathbb{C}[x]$ s'obtient en *raffinant* celle dans $\mathbb{Q}[x]$.



2.1.4 Factoriser pour résoudre

Les logiciels de calcul formel ne permettent pas de calculer la factorisation complète d'un polynôme dans $\mathbb{R}[x]$ ou dans $\mathbb{C}[x]$, sauf à désigner les racines par des lettres (un peu à la façon dont MAPLE contourne le théorème d'Abel).

En 1968, 1970, Berlekamp et Zassenhaus ont mis au point un algorithme de factorisation complète dans $\mathbb{Q}[x]$. Cet algorithme n'est pas très coûteux en temps de calcul et en espace mémoire. Pour fixer les idées, il est nettement moins coûteux que les algorithmes connus de factorisation complète dans $\mathbb{Z}$ mais beaucoup plus que les algorithmes de calcul de pgcd. Il repose sur des techniques d'algèbre pure : à la différence de ce que certains exercices de mathématiques suggèrent, on ne résout pas un polynôme pour le factoriser mais on le factorise dans le but de le résoudre. Il est implanté en MAPLE par la fonction *factor*.

Soit à résoudre un polynôme $P \in \mathbb{Q}[x]$. Une bonne idée consiste donc à tenter de le factoriser dans $\mathbb{Q}[x]$ puis à résoudre chacun de ses facteurs irréductibles. On se ramène donc au problème de la résolution de polynômes irréductibles de $\mathbb{Q}[x]$.

En fait, l'algorithme de résolution que nous allons étudier n'exige pas que le polynôme à résoudre soit irréductible. Il suffit qu'il soit sans facteurs carrés.

Définition 6 *Un polynôme $P \in \mathbb{K}[x]$ est dit* sans facteurs carrés *si tous ses facteurs irréductibles sont simples, c'est–à–dire si sa factorisation complète dans $\mathbb{K}[x]$ s'écrit*

$$P = F_1 \cdots F_r.$$

Proposition 5 *Soient $P \in \mathbb{K}[x]$ un polynôme et P' sa dérivée. Le polynôme $Q = P/(P \wedge P')$ appartient à $\mathbb{K}[x]$, a mêmes racines que P et n'a pas de facteurs carrés.*

Pour calculer Q, il suffit d'un calcul de dérivée (facile), d'un calcul de pgcd (beaucoup plus facile qu'une factorisation complète) et du calcul du quotient d'une division euclidienne (facile).

2.2 Algorithme de Vincent – Collins – Akritas

L'algorithme que nous allons étudier est souvent appelé « algorithme d'Uspensky ». Cette appellation est incorrecte : selon [1], Uspensky s'est attribué, dans la préface de son livre [21], un algorithme dû à Vincent[1] [22]. Cet algorithme a été fortement amélioré par Collins et Akritas [5] et encore plus récemment par Rouillier et Zimmermann [19]. Il s'appuie sur la règle des signes de Descartes [7]. Il isole les racines réelles d'un polynôme sans facteurs carrés.

2.2.1 Intervalles d'isolation

On considère un polynôme $P \in \mathbb{Z}[x]$ sans facteurs carrés. Toutes ses racines sont donc simples. On s'intéresse aux racines réelles positives[2] de P. On cherche une liste d'intervalles $I_1, \ldots, I_m$ qui *isolent* les racines réelles positives de P c'est–à–dire tels que

[1] L'article de Vincent fut publié dès 1834 dans les *Mémoires de la Société Royale de Lille*.

[2] Pour obtenir les racines négatives, il suffit de prendre l'opposé des racines positives du polynôme $P(-x)$.

FIG. 2.3 – Alkiviadis G. Akritas est professeur à l'université de Thessalie, en Grèce.

1. il y ait autant d'intervalles que de racines,
2. il y ait exactement une racine par intervalle,
3. les bornes des intervalles soient des nombres rationnels.

On n'impose pas de précision souhaitée. En effet, les intervalles calculés peuvent facilement être raffinés à volonté, par la méthode dichotomique, en utilisant la proposition suivante.

Proposition 6 *Soient P un polynôme et $a < b$ deux réels. On suppose que l'intervalle $]a,\, b[$ contient zéro ou une racine de P et que $P(a),\, P(b) \neq 0$.*

Alors $]a,\, b[$ isole une racine de P si et seulement si $P(a)\, P(b) < 0$.

2.2.2 Quelques bornes et un algorithme élémentaire

Dans le problème qui nous intéresse comme dans beaucoup d'autres, connaître des bornes permet de mettre au point des algorithmes très simples conceptuellement mais inefficaces. Ces algorithmes servent ensuite de points de comparaison (de bornes en quelque sorte) aux algorithmes plus évolués. La borne C nous servira pour toutes les méthodes. Notons $P = a_d\, x^d + \cdots + a_1\, x + a_0$. On suppose $a_0,\, a_d \neq 0$.

Proposition 7 *Si α est une racine réelle quelconque de P alors*

$$|\alpha| < C(P) \underset{\text{def}}{=} \sum_{i=0}^{d} \left| \frac{a_i}{a_d} \right| .$$

Proposition 8 *Si les coefficients de P sont des entiers et α et β désignent deux racines réelles distinctes de P alors*

$$|\alpha - \beta| > \text{sep}(P) \underset{\text{def}}{=} \frac{1}{d^{\frac{d+2}{2}}\, \|P\|^{d-1}}$$

où $\|P\| = \sqrt{a_d^2 + \cdots + a_0^2}$ désigne la norme euclidienne de P.

Pour pouvoir appliquer la proposition précédente à un polynôme $P \in \mathbb{Q}[x]$ il suffit donc de multiplier P par le plus petit multiple commun des dénominateurs de ses coefficients. Aux lecteurs qui souhaitent en savoir plus on conseille [15] et [2].

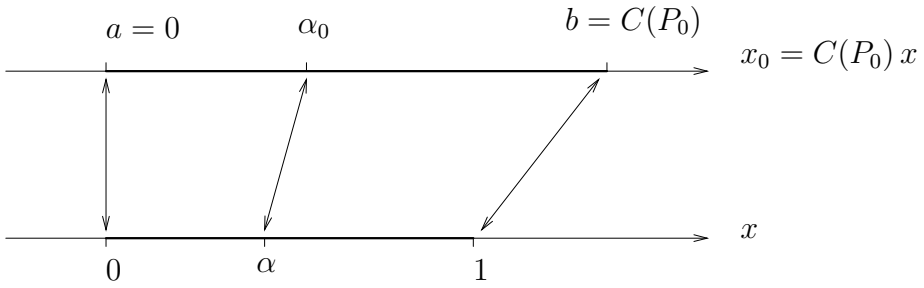
Des deux propositions on déduit un algorithme élémentaire d'isolation des racines réelles positives de P : diviser l'intervalle $]0,\, C(P)[$ en intervalles $I_k =]a_k,\, b_k]$ de longueur inférieure à $\mathrm{sep}(P)$. Chacun de ces intervalles contient au plus une racine de P. Pour décider si l'un d'eux isole une racine, il suffit d'appliquer la proposition 6.

Malheureusement, la borne sep est très pessimiste : l'algorithme obtenu est donc très inefficace. Exemple. Prenons $P = 2\,x^5 - 3\,x + 1$. On a $C(P) = 3$ et $\mathrm{sep}(P) \simeq 10^{-5}$ alors que les racines réelles de P valent approximativement $-1.17,\, 0.33,\, 1$.

2.2.3 L'algorithme de Vincent – Collins – Akritas

Ah si seulement on disposait d'un critère simple qui donnât le nombre de racines de $P(x)$ dans $]0,\, 1[$! On obtiendrait par dichotomie et changement de variable un algorithme qui retournerait un ensemble E d'intervalles isolant les racines réelles positives d'un polynôme $P_0(x_0)$ quelconque (les intervalles étant soit ouverts soit réduits à un point, on le verra). Voici comment.

Les racines en question appartiennent à l'intervalle $]a,\, b[=]0,\, C(P_0)[$. Posons $x_0 = C(P_0)\,x$ et calculons $P(x) \underset{\text{def}}{=} P_0(C(P_0)\,x)$.



Ce changement de variable met en bijection les racines $\alpha \in]0,\, 1[$ de $P(x)$ avec $\alpha_0 \in]a,\, b[$ de $P_0(x_0)$. La bijection est donnée par la formule $\alpha_0 = a + \alpha\,(b - a)$.

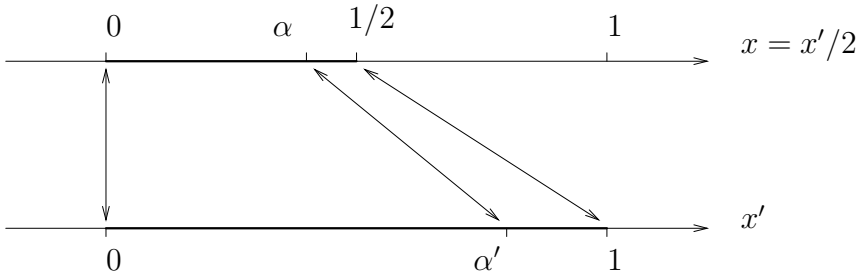
L'exemple suivant illustre le procédé (on a programmé la fonction C). Le changement de variable $x_0 = 12\,x$ envoie la racine $\alpha_0 = 3$ de P_0 sur la racine $\alpha = 1/4$ de P. Il envoie la racine $\alpha_0 = 2$ de P_0 sur la racine $\alpha = 1/6$ de P.

```
P0 := (x0-3)*(x0-2);
                              P0 := (x0 - 3) (x0 - 2)
C(P0);
                                          12

P := subs (x0=12*x, P0);
                            P := (12 x - 3) (12 x - 2)

solve (P);
                                        1/4, 1/6
```

En utilisant notre critère, on peut déterminer le nombre n de racines de $P(x)$ dans $]0,\,1[$ (égal au nombre de racines de $P_0(x_0)$ dans $]a,\,b[$). Si n vaut zéro, E est vide. Si n vaut un, E est réduit au singleton contenant $]a,\,b[$. Si n vaut deux ou plus, on coupe l'intervalle $]0,\,1[$ en deux et on considère séparément les racines de $P(x)$ strictement inférieures à $1/2$ (qui correspondent aux racines de $P_0(x_0)$ strictement inférieures à $(a+b)/2$) de celles strictement supérieures à $1/2$ (qui correspondent aux racines de $P_0(x_0)$ strictement supérieures à $(a+b)/2$). Il se peut bien sûr que $1/2$ soit racine de $P(x)$. Dans ce cas, $(a+b)/2$ est racine de $P_0(x_0)$ et on obtient un intervalle réduit à un point, qu'il faudra ajouter à E. Intéressons–nous au cas des racines de $P(x)$ dans $]0,\,1/2[$. Posons $x = x'/2$ et calculons $Q(x') \underset{\text{def}}{=} P(x'/2)$.



Ce changement de variable met en bijection les racines $\alpha' \in\,]0,\,1[$ de $Q(x')$ avec les racines $\alpha \in\,]0,\,1/2[$ de $P(x)$ et donc avec les racines $\alpha_0 \in\,]a,\,(a+b)/2[$ de $P_0(x_0)$. En notant $]a,\,(a+b)/2[\,=\,]a',\,b'[$, la bijection est donnée par la formule $\alpha_0 = a' + \alpha'\,(b'-a')$. On est donc ramené au problème précédent.

Suite de l'exemple. Le changement de variable (remarquer qu'on a utilisé le même nom de variable pour P et Q) envoie la racine $\alpha = 1/4$ de P (et donc la racine $\alpha_0 = 3$ de P_0) sur la racine $\alpha' = 1/2$ de Q. Il envoie la racine $\alpha = 1/6$ de P (et donc la racine $\alpha_0 = 2$ de P_0) sur la racine $\alpha' = 1/3$ de Q. Le « critère » appliqué à Q indique que P admet deux racines dans $]0,\,1/2[$.

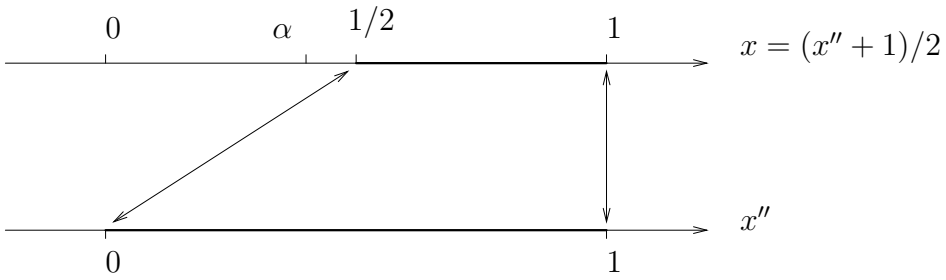
```
P := subs (x0=12*x, P0);
                          P := (12 x - 3) (12 x - 2)

solve (P);
                                      1/4, 1/6

Q := subs (x=x/2, P);
                           Q := (6 x - 3) (6 x - 2)

solve (Q);
                                      1/2, 1/3
```

Intéressons–nous pour finir au cas des racines de $P(x)$ dans $]1/2,\,1[$. Posons $x = (x''+1)/2$ et calculons $R(x'') \underset{\text{def}}{=} P((x''+1)/2)$.



Ce changement de variable met en bijection les racines $\alpha'' \in]0,\, 1[$ de $R(x'')$ avec les racines $\alpha \in]1/2,\, 1[$ de $P(x)$ et donc avec les racines $\alpha_0 \in](a+b)/2,\, b[$ de $P_0(x_0)$. En notant $](a+b)/2,\, b[=]a'',\, b''[$, la bijection est donnée par la formule $\alpha_0 = a'' + \alpha''\,(b'' - a'')$. On est encore ramené au problème précédent.

Suite de l'exemple. Le changement de variable envoie les deux racines de P sur des racines négatives de R. Le « critère » appliqué à R indique que R n'a aucune racine dans $]0,\, 1[$ et donc que P n'a aucune racine dans $]1/2,\, 1[$.

```
P := subs (x0=12*x, P0);
                          P := (12 x - 3) (12 x - 2)

solve (P);
                                    1/4, 1/6

R := subs (x=(x+1)/2, P);
                           R := (6 x + 3) (6 x + 4)

solve (R);
                                  -1/2, -2/3
```

Ne pas oublier de tester si $1/2$ est racine de P. Sur l'exemple, ce n'est pas le cas.

```
subs (x=1/2, P);
                                      12
```

function IsolationRRP ($P_0(x_0)$)
Retourne une liste d'intervalles isolant les racines réelles positives de $P_0(x_0)$
Les intervalles sont soit ouverts soit réduits à un point
begin
  $P(x)$:= $P_0(C(P_0)\,x)$
  return IRRP ($P(x)$, $]0,\, C(P_0)[$)
end

function IRRP ($P(x)$, $]a,\, b[$)
Sous–fonction de la précédente.
Toute racine $\alpha \in]0,\, 1[$ de $P(x)$ correspond à une racine $a + \alpha\,(b-a) \in]a,\, b[$ de $P_0(x_0)$.
begin
  n := le nombre de racines de $P(x)$ dans $]0,\, 1[$ (on utilise le « critère »)
  if $n = 0$ then

```
    return ∅
  elif n = 1 then
    return {]a, b[}
  else
    Q(x') := P(x'/2)
    R(x'') := P((x'' + 1)/2)
    m := (a + b)/2
    if P(1/2) ≠ 0 then
      return IRRP (Q, ]a, m[) ∪ IRRP (R, ]m, b[)
    else
      return IRRP (Q, ]a, m[) ∪ IRRP (R, ]m, b[) ∪ {[m, m]}
    fi
  fi
end
```

Personne ne connaît le « critère » imaginé ci–dessus. On connaît par contre un critère un peu plus faible mais suffisant : la règle des signes de Descartes.

La règle des signes de Descartes

Elle est publiée (sous une forme un peu plus faible) pour la première fois en 1636 dans [7]. Elle fut généralisée dans les années 1820 par Budan et Fourier dans le théorème qui porte aujourd'hui leur nom [2].

Théorème 5 *(règle des signes)*
Soient $P \in \mathbb{R}[x]$ un polynôme, $v(P)$ le nombre de variations de signe dans la liste des coefficients non nuls de P et $r(P)$ le nombre de racines réelles positives de P comptées avec multiplicités. Alors $v(P) \geq r(P)$.

Dans notre cas, le polynôme P est sans carrés et $r(P)$ est donc égal au nombre de racines réelles positives distinctes de P.

D'après la règle des signes, si $v(P) = 0$ alors $r(P) = 0$.

D'après la règle des signes, le théorème des valeurs intermédiaires et le fait que les fonctions polynômes sont continues, si $v(P) = 1$ alors $r(P) = 1$.

Exemple. Le polynôme $P = x^5 - x + 1$ n'a qu'une racine réelle $\alpha = -1.16$. On a $v(P) = 2$ et $r(P) = 0$. Le polynôme $P = 2\,x^5 - 3\,x + 1$ a trois racines réelles $\alpha = -1.17,\ 0.33,\ 1$. On a $v(P) = 2$ et $r(P) = 2$.

Deux problèmes se posent pour pouvoir employer cette règle dans le cadre d'un algorithme d'isolation : d'une part elle ne fournit qu'une borne, d'autre part elle donne des renseignements sur l'intervalle $]0,\ +\infty[$ au lieu de $]0,\ 1[$.

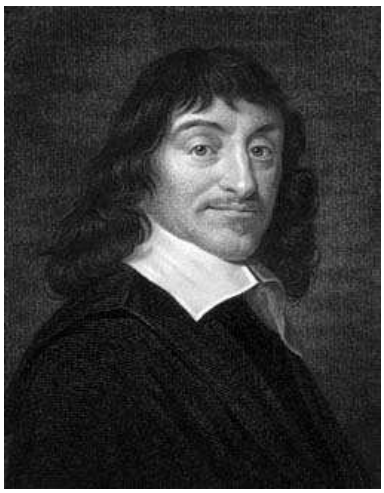


FIG. 2.4 – René Descartes, 1596–1650.

Appliquer la règle des signes dans $]0,\,1[$

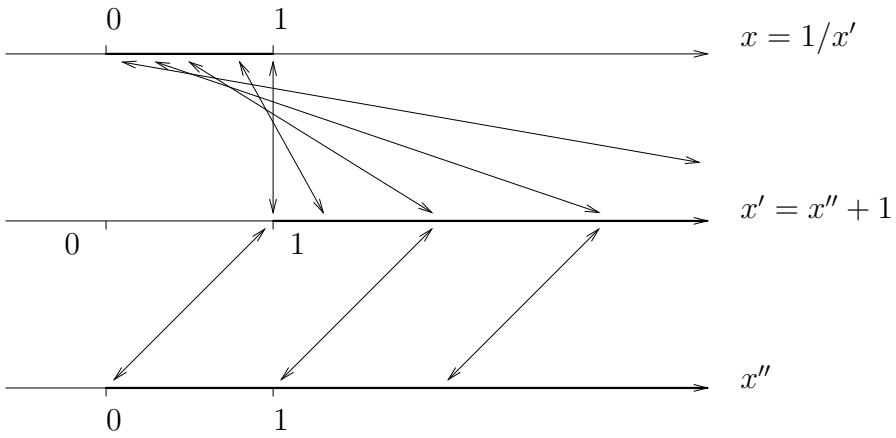
Il suffit de procéder à un changement de variable qui établisse une bijection entre les intervalles $]0,\,1[$ et $]0,\,+\infty[$. Soit $\alpha \in\,]0,\,1[$ racine de $P(x) = a_d\,x^d + \cdots + a_1\,x + a_0$. Posons $x = 1/x'$ et calculons

$$Q(x') \underset{\text{def}}{=} x'^d\,P(1/x').$$

Ce changement de variable met en bijection les racines $\alpha \in\,]0,\,1[$ de P avec les racines $\alpha' \in\,]1,\,+\infty[$ de Q. Posons maintenant $x' = x'' + 1$ et calculons

$$R(x'') \underset{\text{def}}{=} Q(x'' + 1).$$

Ce changement de variable met en bijection les racines $\alpha \in\,]0,\,1[$ de P avec les racines réelles positives de R. Pour majorer le nombre de racines de P dans $]0,\,1[$, il suffit d'appliquer la règle des signes au polynôme R.



On a montré la proposition suivante :

Proposition 9 *Avec les mêmes notations, le nombre $v(R)$ de variations de signes dans la suite des coefficients de R est supérieur ou égal au nombre de racines de P dans l'intervalle $]0,\ 1[$.*

function $v_{01}\,(P(x))$
Retourne une borne supérieure sur le nombre de racines de P dans $]0,\ 1[$
begin
  d := $\deg P$
  $Q(x')$:= $x'^d\,P(1/x')$
  $R(x'')$:= $Q(x''+1)$
  return $v\,(R)$
end

Algorithme de Vincent – Collins – Akritas

On en sait suffisamment pour proposer un algorithme d'isolation des racines réelles positives d'un polynôme fondé sur la règle des signes de Descartes.

La fonction *Vincent_Collins_Akritas* est identique à *isolationRRP*. La fonction *IRRP* est la même que précédemment à la première instruction près. On les rappelle pour mémoire.

function Vincent_Collins_Akritas ($P_0(x_0)$)
Retourne une liste d'intervalles isolant les racines réelles positives de $P_0(x_0)$
Les intervalles sont soit ouverts soit réduits à un point
begin
  $P(x)$:= $P_0(C(P_0)\,x)$
  return IRRP ($P(x)$, $]0,\ C(P_0)[$)
end

function IRRP ($P(x)$, $]a,\ b[$)
Sous–fonction de la précédente.
Toute racine $\alpha \in]0,\ 1[$ de $P(x)$ correspond à une racine $a+\alpha\,(b-a) \in]a,\ b[$ de $P_0(x_0)$.
begin
  n := $v_{01}\,(P)$
  if $n=0$ then
    return $\varnothing$
  elif $n=1$ then
    return $\{]a,\ b[\}$
  else
    $Q(x')$:= $P(x'/2)$
    $R(x'')$:= $P((x''+1)/2)$
    m := $(a+b)/2$
    if $P(1/2)\neq 0$ then
      return IRRP $(Q,\ ]a,\ m[) \cup$ IRRP $(R,\ ]m,\ b[)$
    else

```
        return IRRP (Q, ]a, m[) ∪ IRRP (R, ]m, b[) ∪ {[m, m]}
      fi
    fi
end
```

L'algorithme précédent est correct. Le fait d'utiliser une borne plutôt que le « critère » pose seulement le problème de l'arrêt.

Il est évident que la fonction *IRRP* finit toujours par produire des polynômes P ayant zéro ou une racine dans $]0,\ 1[$ mais il n'est pas évident que le nombre de variations de signes dans la suite des coefficients de ces polynômes (ou plutôt des polynômes R calculés dans v_{01}) vaille zéro ou un.

On pourrait compliquer un peu l'algorithme en utilisant la proposition 6 dès que l'intervalle $]a,\ b[$ est de longueur inférieure à $\mathrm{sep}(P_0)$. On déciderait ainsi si P_0 admet zéro ou une racine dans $]a,\ b[$ en testant le signe de $P(0)\,P(1)$.

En fait, Vincent a démontré que cette complication était inutile et que l'algorithme s'arrêtait dans tous les cas.

Plus précisément, il a montré que si $]0,\ 1[$ contient zéro (resp. une) racine de P et si toutes les racines complexes de P sont « suffisamment éloignées » de l'origine alors $v(P) = 0$ (resp. $v(P) = 1$). Ce cas finit toujours par se produire parce que le « zoom » effectué par l'algorithme éloigne les racines de P les unes des autres et les éloigne de l'origine. Voir [19, 2, théorème des deux cercles] pour une formulation précise.

Implantation

L'algorithme de Vincent – Collins – Akritas est très efficace parce que les trois changements de variables qu'il effectue sont très faciles à implanter (cf. exercices). Il satisfait les spécifications énoncées en début de section. En particulier, les bornes des intervalles sont des rationnels. En MAPLE, il est disponible sous le nom *realroot*.

```
readlib (realroot);
                    proc(poly, widthgoal)  ...  end proc
```

2.3 Éléments d'arithmétique par intervalles

Une fois calculé un intervalle isolant une solution d'une équation $P(x_1) = 0$, il reste dans la chaîne GB + RS à calculer l'image de cet intervalle par des fonctions polynômes (les fonctions F_i qui donnent les valeurs des coordonnées $x_2, \ldots, x_n$ de la solution du système en fonction de la valeur de x_1). C'est ce problème qui est considéré dans cette section.

L'arithmétique par intervalle est une discipline à part entière. Elle a été inventée dans les années 50 pour contrôler les erreurs d'arrondis. Elle a été généralisée dans les années 60 pour contrôler les erreurs de toute nature (arrondis, mesure, incertitude ...). Voir [11].

Un intervalle $X = [a,\ b]$ représente un nombre $a \leq \alpha \leq b$ connu avec incertitude. Tout nombre a peut être converti en un intervalle $[a,\ a]$.

Soient $X = [a,\, b]$ et $Y = [c,\, d]$ et op une opération arithmétique élémentaire. On définit

$$X \,\mathrm{op}\, Y = \{x \,\mathrm{op}\, y \mid x \in X,\, y \in Y\}.$$

Si les bornes sont exactes et op $= +,\, -,\, \times$ (notre cas) $X \,\mathrm{op}\, Y$ peut se calculer facilement :

$$[a,\, b] + [c,\, d] = [a + c,\, b + d]$$

$$[a,\, b] - [c,\, d] = [a - d,\, b - c]$$

$$[a,\, b] \times [c,\, d] = [\min(ac,\, ad,\, bc,\, bd),\, \max(ac,\, ad,\, bc,\, bd)]$$

Dans le cas d'intervalles dotés de bornes à virgule flottante, des problèmes spécifiques se posent qui empêchent de satisfaire les spécifications ci–dessus. Ces problèmes ne nous concernent pas.

2.3.1 Le problème des dépendances

On est amené à considérer des systèmes de la forme

$$\begin{cases} y = F(x), \\ P(x) = 0. \end{cases}$$

Les intervalles X qui isolent les racines réelles de p ont des bornes exactes. La fonction polynôme

$$F(x) = a_d\, x^d + \cdots + a_1\, x + a_0$$

ne met en œuvre que des opérations op $\in \{+,\, -,\, \times\}$. Définissons la fonction $\mathscr{F}(X)$ par

$$\begin{array}{rcl} \mathscr{F}(X) & = & a_d \times X^d + \cdots + a_1 \times X + a_0 \\ X^k & = & X \times \cdots \times X \quad (k \text{ fois}) \end{array}$$

On pourrait penser que

$$\mathscr{F}(X) = F(X) \underset{\text{def}}{=} \{F(x) \mid x \in X\}.$$

Malheureusement c'est faux ! Prenons par exemple $F(x) = x^2$ et $X = [\,-1,\, 1]$. On a $F(X) = [0,\, 1]$ mais $\mathscr{F}(X) = X \times X = [\min(-1,\, 1),\, \max(-1,\, 1)] = [\,-1,\, 1]$. Par définition on a

$$[\,-1,\, 1] \times [\,-1,\, 1] = \{x\, y \mid -1 \leq x \leq 1,\, -1 \leq y \leq 1\}.$$

Les intervalles représentent deux nombres indépendants alors que nous voudrions qu'ils représentent deux nombres dépendants (ici, une même racine de P). En d'autres mots nous voudrions que

$$[\,-1,\, 1] \times [\,-1,\, 1] = \{x\, x \mid -1 \leq x \leq 1\}.$$

Dans le cas de la fonction « puissance » le problème peut être résolu mais le problème des dépendances se repose lors du calcul de la somme des monômes : les monômes de degré strictement positif représentent des nombres différents mais dépendants deux à deux. Sauf à étudier

formellement $F(x)$ sur l'intervalle X le problème des dépendances ne peut pas être complètement résolu et on doit se contenter de la spécification :

$$\mathscr{F}(X) \supset F(X).$$

N'importe comment, cette difficulté est une fausse difficulté pour RS puisque l'intervalle X peut être raffiné à volonté (proposition 6). D'où l'intérêt pédagogique d'étudier les algorithmes « en situation ».

2.4 Exemple

On montre sur un exemple toutes les techniques développées dans cette partie. On s'intéresse aux solutions $(x,\, y) \in \mathbb{R}^2$ du système $\mathscr{S}$ suivant (avec $x > 0$).

$$x^3 - 4\,x^2 + 5\,x - 2 = 0, \quad y = x^2 + x + 1.$$

Le polynôme $x^3 - 4\,x^2 + 5\,x - 2 = (x-1)^2\,(x-2)$ a deux racines évidentes mais on fait bien sûr comme si on ne le savait pas.

2.4.1 Suppression des multiplicités

On commence par calculer le pgcd de $R_0 = x^3 - 4\,x^2 + 5\,x - 2$ et de sa dérivée. On applique l'algorithme d'Euclide dans $\mathbb{Q}[x]$. Le pgcd est le dernier reste non nul : R_2.

```
R [0] := expand ((x-1)^2*(x-2));
                                  3      2
                        R[0] := x  - 4 x  + 5 x - 2

R [1] := diff (R [0], x);
                                    2
                          R[1] := 3 x  - 8 x + 5

R [2] := rem (R [0], R [1], x);
                                          2 x
                             R[2] := 2/9 - ---
                                           9

R [3] := rem (R [1], R [2], x);
                                  R[3] := 0
```

Le polynôme $P_0 = R_0/R_2$ a mêmes racines que R_0 et n'a que des racines simples. Simplifier ses coefficients ne change pas ses racines.

```
P0 := quo (R [0], R [2], x);
                                    2
                        P0 := -9/2 x  + 27/2 x - 9

P0 := P0 / lcoeff (P0, x);
                                  2
                          P0 := x  - 3 x + 2
```

On est donc ramené à l'étude des racines réelles positives de $P_0 = x^2 - 3\,x + 2$.

2.4.2 Isolation des racines réelles positives

On a $C(P_0) = 6$. Les racines réelles positives de P_0 appartiennent à l'intervalle $]a,\, b[=]0,\, 6[$. Ces racines sont en bijection avec les racines du polynôme $P(x) = P_0(6\,x)$ appartenant à l'intervalle $]0,\, 1[$. Dans les calculs ci–dessous on a programmé la fonction v mentionnée dans la règle des signes.

```
P := expand (subs (x=6*x, P0));
                                  2
                       P := 36 x  - 18 x + 2
```

On calcule $v_{01}(P)$. On trouve 2.

```
d := degree (P, x);
                             d := 2

Q := expand (x^d * subs (x = 1/x, P));
                                                2
                       Q := 36 - 18 x + 2 x

R := expand (subs (x = x + 1, Q));
                                                2
                       R := 20 - 14 x + 2 x

v (R);
                                      2
```

On vérifie rapidement que $1/2$ n'est pas racine de P.

```
subs (x = 1/2, P);
                                      2
```

On s'intéresse aux racines de P appartenant à l'intervalle $]0,\, 1/2[$. On calcule par changement de variable un nouveau polynôme P_a. Les racines de P_a appartenant à $]0,\, 1[$ sont en bijection avec les racines de P_0 appartenant à $]0,\, 3[$.

```
Pa := expand (subs (x = x/2, P));
                                  2
                       Pa := 9 x  - 9 x + 2
```

On calcule $v_{01}(P_a)$. Inutile de recalculer d. On trouve encore 2.

```
Qa := expand (x^d * subs (x = 1/x, Pa));
                                  2
                       Qa := 2 x  - 9 x + 9

Ra := expand (subs (x = x + 1, Qa));
                                  2
                       Ra := 2 x  - 5 x + 2

v (Ra);
                                    2
```

On vérifie que $1/2$ n'est pas racine de P_a.

```
subs (x = 1/2, Pa);
                                    -1/4
```

On s'intéresse aux racines de P_a appartenant à l'intervalle $]0,\ 1/2[$. On calcule par changement de variable un nouveau polynôme P_{aa}. Les racines de P_{aa} appartenant à $]0,\ 1[$ sont en bijection avec les racines de P_0 appartenant à $]0,\ 3/2[$.

```
Paa := expand (subs (x = x / 2, Pa));
                                      2
                          Paa := 9/4 x  - 9/2 x + 2
```

On calcule $v_{01}(P_{aa})$. On trouve 1.

```
Qaa := expand (x^d * subs (x = 1/x, Paa));
                                    2
                          Qaa := 2 x  - 9/2 x + 9/4

Raa := expand (subs (x = x + 1, Qaa));
                                    2
                          Raa := 2 x  - 1/2 x - 1/4

v (Raa);
                                      1
```

On en déduit que l'intervalle $]0,\ 3/2[$ isole une racine de P_0.

On s'intéresse aux racines de P_a appartenant à l'intervalle $]1/2,\ 1[$. On calcule par changement de variable un nouveau polynôme P_{ab}. Les racines de P_{ab} appartenant à $]0,\ 1[$ sont en bijection avec les racines de P_0 appartenant à $]3/2,\ 3[$.

```
Pab := expand (subs (x = (x+1)/2, Pa));
                                       2
                            Pab := 9/4 x  - 1/4
```

On calcule $v_{01}(P_{ab})$. On trouve 1.

```
Qab := expand (x^d * subs (x = 1/x, Pab));
                                     2
                           Qab := - 1/4 x  + 9/4

Rab := expand (subs (x = x + 1, Qab));
                                      2
                          Rab := - 1/4 x  - 1/2 x + 2

v (Rab);
                                      1
```

On en déduit que l'intervalle $]3/2,\ 3[$ isole une racine de P_0.

Un polynôme de degré deux admet au plus deux racines réelles. On pourrait donc s'arrêter ici mais par acquis de conscience, on vérifie que P_0 n'admet aucune racine dans l'intervalle $]3,\ 6[$.

```
Pb := expand (subs (x = (x + 1)/2, P));
                                  2
                          Pb := 9 x  + 9 x + 2

Qb := expand (x^d * subs (x = 1/x, Pb));
                                  2
                          Qb := 2 x  + 9 x + 9

Rb := expand (subs (x = x + 1, Qb));
                                  2
                         Rb := 2 x  + 13 x + 20

v (Rb);
                                  0
```

En conclusion, P_0 admet deux racines réelles, isolées dans les intervalles ci–dessous.

$$\left]0,\,\frac{3}{2}\right[,\quad \left]\frac{3}{2},\,3\right[.$$

2.4.3 Report des racines

On a isolé les abscisses x des deux solutions. On cherche à en encadrer les ordonnées y. On reporte les deux intervalles calculés dans la fonction

$$F(x) = x^2 + x + 1.$$

On choisit pour effectuer les calculs la fonction

$$\mathscr{F}(X) = X \times X + X + 1.$$

On trouve pour le premier intervalle

$$\left]0,\,\frac{3}{2}\right[\times\left]0,\,\frac{3}{2}\right[+\left]0,\,\frac{3}{2}\right[+1=\left]0,\,\frac{9}{4}\right[+\left]0,\,\frac{3}{2}\right[+1=\left]0,\,\frac{15}{4}\right[+1=\left]1,\,\frac{19}{4}\right[.$$

On trouve pour le deuxième

$$\left]\frac{3}{2},\,3\right[\times\left]\frac{3}{2},\,3\right[+\left]\frac{3}{2},\,3\right[+1=\left]\frac{9}{4},\,9\right[+\left]\frac{3}{2},\,3\right[+1=\left]\frac{15}{4},\,12\right[+1=\left]\frac{19}{4},\,13\right[.$$

On en déduit que les solutions recherchées du système $\mathscr{S}$ appartiennent aux « boîtes » suivantes :

$$(x,\,y) = \left(\left]0,\,\frac{3}{2}\right[,\,\left]1,\,\frac{19}{4}\right[\right),\quad (x,\,y) = \left(\left]\frac{3}{2},\,3\right[,\,\left]\frac{19}{4},\,13\right[\right).$$

2.4.4 Amélioration de la précision

Supposons qu'on souhaite améliorer la précision de la deuxième boîte (l'intervalle encadrant l'ordonnée est de largeur supérieure à 8). On procède par dichotomie.

```
m := (3/2 + 3) / 2;
                                    m := 9/4

subs (x = 3/2, P0), subs (x = m, P0), subs (x = 3, P0);
                                  -1/4, 5/16, 2
```

On conclut que l'abscisse de la deuxième solution de $\mathscr{S}$ appartient en fait à l'intervalle

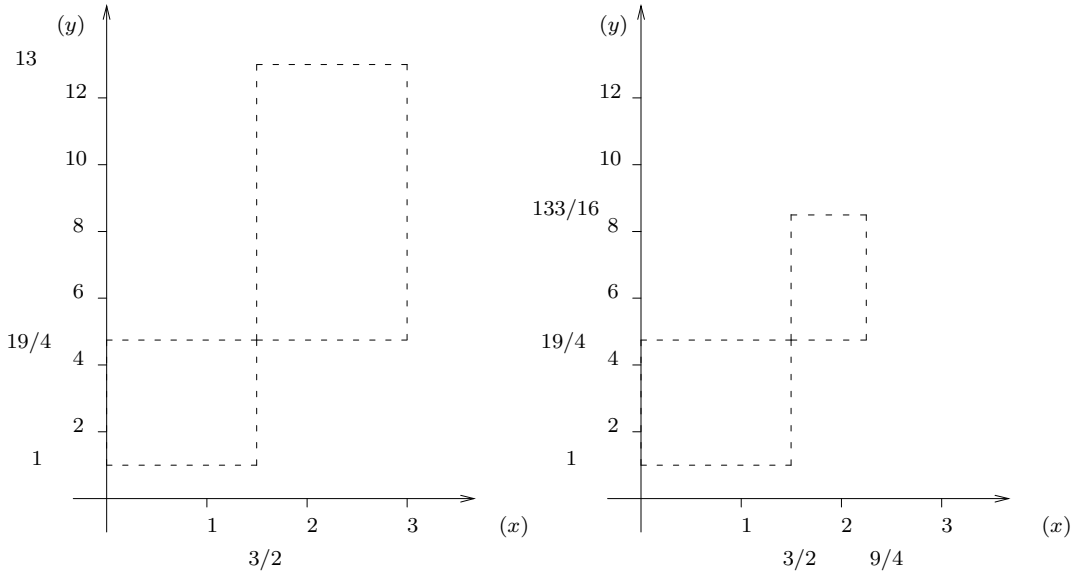
$$\left]\frac{3}{2}, \frac{9}{4}\right[.$$

On recommence le calcul d'un intervalle encadrant l'ordonnée

$$\left]\frac{3}{2}, \frac{9}{4}\right[\times \left]\frac{3}{2}, \frac{9}{4}\right[+ \left]\frac{3}{2}, \frac{9}{4}\right[+ 1 = \left]\frac{9}{4}, \frac{91}{16}\right[+ \left]\frac{3}{2}, \frac{9}{4}\right[+ 1 = \left]\frac{15}{4}, \frac{117}{16}\right[+ 1 = \left]\frac{19}{4}, \frac{133}{16}\right[.$$

L'intervalle encadrant l'ordonnée est maintenant de largeur inférieure à 4.

Boîtes encadrant les solutions de $\mathscr{S}$



Avant amélioration

Après amélioration

Chapitre 3

Simplification de systèmes

Dans cette partie, on s'intéresse au problème de la simplification (en un certain sens) de systèmes d'équations polynomiales en plusieurs indéterminées $x_1, \ldots, x_n$. La simplification a pour but de transformer le système d'entrée $\mathscr{S}$ en un système équivalent $\mathscr{G}$ (ayant mêmes solutions) de la forme

$$P(x_1) = 0,\ x_2 = F_2(x_1),\ \ldots,\ x_n = F_n(x_1)$$

où P et les F_i sont des polynômes de $\mathbb{Q}[x_1]$. Ainsi, résoudre $\mathscr{S}$ revient à résoudre $P(x_1) = 0$ puis à reporter les racines obtenues dans les polynômes F_i pour obtenir les autres coordonnées des solutions (ce qu'on a vu dans le chapitre précédent). Par exemple le système $\mathscr{S}$ suivant, qui décrit l'intersection d'une droite et d'un cercle, peut être transformé en le système équivalent $\mathscr{G}$ (substituer x à y dans (S_1)).

$$\mathscr{S}\begin{cases} (S_2) & x - y = 0, \\ (S_1) & x^2 + y^2 - 1 = 0. \end{cases} \qquad \mathscr{G}\begin{cases} (G_2) & y = x, \\ (G_1) & 2\,x^2 - 1 = 0. \end{cases}$$

Pour résoudre $\mathscr{S}$, il suffit de résoudre l'équation (G_1) puis de reporter les deux valeurs de x dans (G_2). La transformation souhaitée n'est pas toujours possible. Elle est impossible dans le cas où $\mathscr{S}$ a une infinité de solutions dans $\mathbb{C}^n$ par exemple. Même dans le cas où elle est possible théoriquement, le calcul échoue souvent en pratique : le système simplifié peut être beaucoup plus volumineux que le système $\mathscr{S}$. L'algorithme de simplification est dû à Buchberger [4]. Il produit en sortie ce qu'on appelle une « base de Gröbner » de l'idéal engendré par le système $\mathscr{S}$. La version implantée dans le logiciel GB [9] est nettement plus évoluée que celle que nous présentons.

3.1 Solutions, idéaux

On considère un système fini $\mathscr{S}$ d'un anneau de polynômes $\mathbb{A} = \mathbb{Q}[x_1, \ldots, x_n]$ en plusieurs indéterminées à coefficients rationnels.

Abus de langage 1 *On s'autorise à confondre un ensemble de polynômes avec le système d'équations qui lui correspond.*

3.1.1 Solutions d'un système

Définition 7 *(solution)*
On appelle solution *de $\mathscr{S}$ tout n–uplet $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{C}^n$ de nombres complexes qui annule tous les éléments de $\mathscr{S}$ (c'est–à–dire tel que $P(\alpha) = 0$ quel que soit $P \in \mathscr{S}$).*

Définition 8 *(solution réelle)*
Une solution $(\alpha_1, \ldots, \alpha_n)$ *de $\mathscr{S}$ est dite* réelle *si toutes ses composantes le sont, c'est–à–dire si $(\alpha_1, \ldots, \alpha_n) \in \mathbb{R}^n$.*

Exemple. Le système $\mathscr{S}$ donné en introduction a deux solutions réelles : $(\sqrt{2}/2,\, \sqrt{2}/2)$ et $(-\sqrt{2}/2,\, -\sqrt{2}/2)$.

3.1.2 Idéal engendré

Définition 9 *(idéal)*
On appelle idéal *de $\mathbb{A}$ tout sous–ensemble non vide $\mathscr{I}$ de $\mathbb{A}$ stable par addition interne et par multiplication par un élément de $\mathbb{A}$.*

En d'autres termes, dire que $\mathscr{I}$ est un idéal de $\mathbb{A}$ c'est dire que

$$A,\, B \in \mathscr{I} \Rightarrow A + B \in \mathscr{I}, \quad A \in \mathscr{I},\, B \in \mathbb{A} \Rightarrow A\, B \in \mathscr{I}.$$

Exemples. L'ensemble $\{0\}$ est un idéal de $\mathbb{A}$. L'anneau $\mathbb{A}$ est aussi un idéal de $\mathbb{A}$. L'ensemble des nombres pairs est un idéal de $\mathbb{Z}$ (la somme de deux nombres pairs est un nombre pair ; le produit d'un nombre pair par un nombre quelconque est encore un nombre pair).

Proposition 10 *L'intersection d'une famille d'idéaux de $\mathbb{A}$ est encore un idéal de $\mathbb{A}$.*

Définition 10 *On appelle* idéal engendré par *$\mathscr{S}$ dans $\mathbb{A}$ l'intersection de tous les idéaux de $\mathbb{A}$ qui contiennent S. C'est aussi le plus petit idéal $\mathscr{I}$ de $\mathbb{A}$ contenant $\mathscr{S}$. L'ensemble $\mathscr{S}$ est appelé une* base *de $\mathscr{I}$.*

Proposition 11 *L'idéal engendré par $\mathscr{S} = \{S_1, \ldots, S_m\}$ dans $\mathbb{A}$ est l'ensemble de toutes les combinaisons linéaires des éléments de $\mathscr{S}$ avec des éléments quelconques de $\mathbb{A}$ pour coefficients :*

$$\mathscr{I} = \{A_1\, S_1 + \cdots + A_m\, S_m \mid A_i \in \mathbb{A}\}.$$

Exemple. Les systèmes $\mathscr{S} = \{x^2 + y^2 - 1,\, x - y\}$ et $\mathscr{G} = \{2\, x^2 - 1,\, x - y\}$ donnés en introduction engendrent le même idéal dans l'anneau $\mathbb{A} = \mathbb{Q}[x, y]$ (ce sont deux bases du même idéal). Pour le montrer, il suffit de montrer que tout élément de $\mathscr{S}$ appartient à l'idéal engendré par $\mathscr{G}$ et réciproquement. Effectivement,

$$\underbrace{x^2 + y^2 - 1}_{S_1} + (x + y)\,\underbrace{(x - y)}_{S_2} = \underbrace{2\, x^2 - 1}_{G_1}, \qquad \underbrace{2\, x^2 - 1}_{G_1} - (x + y)\,\underbrace{(x - y)}_{G_2} = \underbrace{x^2 + y^2 - 1}_{S_1}.$$

Exemple. Dans $\mathbb{Z}$ l'idéal engendré par 5 est l'ensemble de tous les entiers multiples de 5.

$$(5) = 5\,\mathbb{Z} = \{0,\, -5,\, 5,\, -10,\, 10,\, -15,\, \ldots\}.$$

Définition 11 *(équivalence modulo un idéal)*
Soient P, $Q \in \mathbb{A}$ et $\mathscr{I}$ un idéal de $\mathbb{A}$. On dit que P est équivalent à Q modulo $\mathscr{I}$*, ce qu'on note $P \equiv Q \mod \mathscr{I}$ si $P - Q \in \mathscr{I}$.*

Exemple. Dans $\mathbb{Z}$ on a $3 \equiv 13 \mod (5)$.

Proposition 12 *(les équivalences modulo un idéal se comportent comme des égalités)*
Soient P, Q, P', $Q' \in \mathbb{A}$ et $\mathscr{I}$ un idéal de $\mathbb{A}$. Si $P \equiv Q \mod \mathscr{I}$ et $P' \equiv Q' \mod \mathscr{I}$ alors

$$P + P' \equiv Q + Q' \mod \mathscr{I}, \qquad P \times P' \equiv Q \times Q' \mod \mathscr{I}.$$

Exemple. Dans $\mathbb{Z}$, modulo l'idéal (5) on a (en notant $=$ pour $\equiv$)

$$\begin{array}{ccc} 1 & = & 6 \\ + & & + \\ 3 & = & 13 \\ \| & & \| \\ 4 & = & 19 \end{array} \qquad \begin{array}{ccc} 1 & = & 6 \\ \times & & \times \\ 3 & = & -2 \\ \| & & \| \\ 3 & = & -12 \end{array}$$

Exemple. On a $x^3 \equiv y^3 \mod \mathscr{I}$ où $\mathscr{I}$ désigne l'idéal de $\mathbb{A} = \mathbb{Q}[x, y]$ engendré par le système $\mathscr{S}$ donné en introduction. C'est évident d'après la proposition précédente : comme $x - y \in \mathscr{I}$ on a $x \equiv y \mod \mathscr{I}$ et donc $x \times x \times x \equiv y \times y \times y \mod \mathscr{I}$. Autre preuve : $x^3 - y^3 = (x - y)(x^2 + xy + y^2)$. Comme $x - y \in \mathscr{I}$ on a $x^3 - y^3 \in \mathscr{I}$.

3.1.3 Correspondance entre solutions d'un système et idéal engendré

Proposition 13 *Soient P un polynôme appartenant à l'idéal $\mathscr{I}$ engendré par $\mathscr{S} = \{S_1, \ldots, S_m\}$ dans $\mathbb{A}$. Toute solution de $\mathscr{S}$ est solution de P.*

Preuve Comme $P \in \mathscr{I}$, il existe des polynômes $A_1, \ldots, A_m$ tels que $P = A_1 S_1 + \cdots + A_m S_m$. Une solution α de $\mathscr{S}$ est un n–uplet de nombres complexes tels que $S_1(\alpha) = \cdots = S_m(\alpha) = 0$. Par conséquent $P(\alpha) = A_1(\alpha) S_1(\alpha) + \cdots + A_m(\alpha) S_m(\alpha) = 0$. □

La proposition précédente a plusieurs conséquences :
– si deux polynômes P et Q sont équivalents modulo $\mathscr{I}$ alors $P(\alpha) = Q(\alpha)$ pour toute solution α de $\mathscr{S}$;
– deux systèmes qui engendrent le même idéal sont équivalents (ils ont mêmes solutions).

Exemple. Les systèmes $\mathscr{S}$ et $\mathscr{G}$ donnés en introduction engendrent le même idéal. Ils ont mêmes solutions.

Exemple. Les polynômes x^3 et y^3 sont équivalents modulo l'idéal $\mathscr{I}$ engendré par le système donné en introduction. On peut comprendre cela ainsi :

$$\forall\, (x,\, y) \in \left\{ \left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2} \right), \left(-\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2} \right) \right\}, \quad x^3 = y^3.$$

3.2 Ordres admissibles

3.2.1 Termes, monômes

Définition 12 *(terme, monôme)*

On appelle terme *sur l'alphabet* $\{x_1, \ldots, x_n\}$ *tout produit de puissances* $x_1^{a_1} \cdots x_n^{a_n}$*. Le terme* $x_1^0 \cdots x_n^0$ *est noté* 1.

Un monôme *est le produit d'un terme par un élément non nul de* $\mathbb{Q}$.

On définit le degré total *d'un terme* $t = x_1^{a_1} \cdots x_n^{a_n}$ *par* $\deg t = a_1 + \cdots + a_n$.

Tout polynôme P est donc une somme de monômes (ou une combinaison linéaire de termes) :

$$P = c_1\, t_1 + \cdots + c_s\, t_s.$$

Par exemple, $3\, x_1\, x_2 + 7$ est vu comme $c_1\, t_1 + c_2\, t_2$ avec $c_1 = 3$ et $t_1 = x_1\, x_2$ et $c_2 = 7$ et $t_2 = 1$.

3.2.2 Ordres admissibles

L'utilité des ordres admissibles apparaîtra dans la section suivante.

Définition 13 *(ordre admissible)*

Un ordre total sur l'ensemble des termes est dit admissible *s'il satisfait les propriétés suivantes pour tous termes* t*,* t' *et* t'' *:*

1. $1 \leq t$
2. $t < t' \Rightarrow t\, t'' < t'\, t''$

Dans le cas d'une seule indéterminée x, il n'y a qu'un seul ordre admissible :

$$1 < x < x^2 < x^3 < \cdots$$

Définition 14 *(ordre lexicographique défini par* $x_n > \cdots > x_1$*)*

Soient $t = x_1^{a_1} \cdots x_n^{a_n}$ *et* $t' = x_1^{a'_1} \cdots x_n^{a'_n}$ *deux termes. On pose que* $t < t'$ *si il existe un indice* $1 \leq i \leq n$ *tel que* $a_i < a'_i$ *et* $a_j = a'_j$ *pour tous* $i < j \leq n$*.*

Pour comparer t et t', on peut procéder ainsi : on écrit les n–uplets d'exposants $(a_1, \ldots, a_n)$ et $(a'_1, \ldots, a'_n)$ puis on compare les exposants deux–à–deux en allant de la droite vers la gauche (on commence par comparer a_n et a'_n etc. jusqu'à a_1 et a'_1). On s'arrête à la première différence rencontrée. Mettons que ce soit à l'indice i. Alors $t < t'$ si $a_i < a'_i$.

Les ordres lexicographiques sont des ordres admissibles. Ils correspondent à « l'ordre du dictionnaire » (il y en a plusieurs parce qu'il y a plusieurs façons d'ordonner l'alphabet des indéterminées).

Exemple. Voici un « extrait » de l'ordre lexicographique défini par $y > x$

$$1 < x < x^2 < x^3 < \cdots < y < yx < yx^2 < yx^3 < \cdots < y^2 < y^2x < \cdots$$

Définition 15 *(ordre compatible avec le degré total)*

Un ordre admissible est dit compatible avec le degré total *s'il satisfait, outre les deux propriétés des ordres admissibles, la propriété*

3. $\deg t < \deg t' \Rightarrow t < t'$ *pour tous termes* $t,\ t'$*.*

Exemple. Voici un extrait d'un ordre compatible avec le degré total (on ordonne les termes de même degré total entre eux avec l'ordre lexicographique défini par $y > x$)

$$1 < x < y < x^2 < xy < y^2 < x^3 < x^2y < xy^2 < y^3 < x^4 < \cdots$$

On peut définir des ordres admissibles « lexicographiques par blocs ». Pour cela, considérons deux alphabets $x_1, \ldots, x_n$ et $y_1, \ldots, y_m$. Tout terme t sur l'union de ces deux alphabets peut être décomposé de façon unique en un produit $t = t_x\, t_y$ d'un terme t_x sur l'alphabet des x et d'un terme t_y sur l'alphabet des y.

Définition 16 *(ordre lexicographique par blocs)*

Avec les mêmes notations. Supposons les deux alphabets ordonnés chacun suivant un ordre amissible. On définit l'ordre lexicographique par blocs

$$(x_1, \ldots, x_n) \gg (y_1, \ldots, y_m)$$

par : $t < t'$ *si* $t_x < t'_x$ *ou (*$t_x = t'_x$ *et* $t_y < t'_y$*).*

3.3 Réécriture

Définition 17 *(règle de réécriture)*

Une règle de réécriture *est une règle de substitution de la forme*

$$\text{monôme} \longrightarrow \text{polynôme}.$$

Le monôme en partie gauche est appelé monôme de tête *de la règle. Le terme de ce monôme est appelé le* terme de tête *de la règle.*

Définition 18 *Un* système de réécriture *est un ensemble de règles de réécriture.*

Convention 1 *On détermine le terme de tête d'un polynôme* P *en fixant un ordre admissible sur l'ensemble des termes : le terme de tête de* P *est le plus grand terme figurant dans* P *pour l'ordre admissible choisi.*

Le procédé ci–dessus peut paraître compliqué : pourquoi ne pas tout simplement indiquer le terme de tête qu'on souhaite pour P ? Parce que certains algorithmes vont devoir déterminer le terme de tête de polynômes qui n'apparaissent qu'au cours des calculs. Il faut qu'ils soient capables de déterminer le terme de tête d'un polynôme quelconque. Pourquoi exige–t–on que les ordres soient admissibles ? Pour éviter des systèmes de réécriture « pathologiques » qui pourraient conduire à des réécritures infinies tels que

$$x \longrightarrow x^2 \quad \text{ou} \quad x\,y \longrightarrow z, \quad x\,z \longrightarrow x^2\,y.$$

Proposition 14 *Soit $\mathscr{S}$ un système de réécriture dont les monômes de tête sont donnés par un ordre admissible. Toute réécriture par $\mathscr{S}$ est finie.*

La fonction `leadterm` du paquetage `Groebner` de MAPLE est paramétrée par un polynôme et un ordre admissible. Elle retourne le terme de tête du polynôme pour l'ordre admissible. Le paramètre `plex(y,x)` désigne l'ordre lexicographique défini par $y > x$. Le paramètre `tdeg(y,x)` désigne l'ordre compatible avec le degré total donné en exemple ci–dessus. Suivant l'ordre admissible le terme de tête de P est y^3 ou $x^2\,y^2$. Le terme x ne peut pas être le terme de tête de P puisque, quel que soit l'ordre admissible, $x^2\,y^2 > x$.

```
with (Groebner):
P := y^3 + 3*x^2*y^2 + x;
```

$$P := y^3 + 3\,x^2\,y^2 + x$$

```
leadterm (P, plex(y,x));
```

$$y^3$$

```
leadterm (P, tdeg(y,x));
```

$$x^2\,y^2$$

Une fois fixé un ordre admissible tout système d'équations peut être vu comme un système de réécriture et réciproquement.

Abus de langage 2 *Une fois fixé un ordre admissible, on s'autorise à confondre un ensemble de polynômes $\mathscr{S}$ avec le système d'équations $\mathscr{S} = 0$ et le système de réécriture qu'il définit.*

Définition 19 *(réduction)*

Soient $m \longrightarrow Q$ une règle de réécriture, $S = m - Q$ le polynôme correspondant à la règle et $P = m_1 + \cdots + m_r$ un polynôme.

On dit que P est réductible *par la règle si l'un des monômes m_i est divisible par m, c'est–à–dire s'il existe un indice $1 \leq i \leq r$ et un monôme m'_i tels que $m_i = m\,m'_i$.*

On dit que le polynôme $Q = (P - m'_i\,S)$ s'obtient en réduisant P une fois par S. On note $P \xrightarrow[S]{} Q$.

Définition 20 *Soient $\mathscr{S}$ un système de réécriture, P et Q deux polynômes. On dit que P est* réductible *par $\mathscr{S}$ s'il existe $S \in \mathscr{S}$ tel que P soit réductible par S, ce qu'on note $P \xrightarrow[\mathscr{S}]{} Q$.*

Exemple. Le système $\mathscr{S} = \{x^2+y^2-1,\ x-y\}$ peut être transformé en le système de réécriture (en fixant l'ordre lexicographique défini par $y > x$) :

$$(S_1)\ y^2 \longrightarrow 1 - x^2, \qquad (S_2)\ y \longrightarrow x.$$

Le polynôme y^3 peut se réécrire de plusieurs façons différentes :

$$(y - x^2\, y) \xleftarrow[S_1]{} y^3 \xrightarrow[S_2]{} x\, y^2.$$

Définition 21 *La notation $P \xrightarrow[\mathscr{S}]{*} Q$ signifie que P se réécrit en Q en appliquant un nombre quelconque (éventuellement nul) de fois des règles de $\mathscr{S}$.*

Proposition 15 *(on réécrit un polynôme en un polynôme équivalent)*
Soient $\mathscr{S}$ un système de réécriture et $\mathscr{I}$ l'idéal qu'il engendre. Si $P \xrightarrow[\mathscr{S}]{*} Q$ alors $P \equiv Q \mod \mathscr{I}$.

Suite de l'exemple. Notons $P = y^3$ et $Q = y - x^2\, y$. On vérifie facilement que $P = Q - y\, S_1$ et donc que $P \equiv Q \mod \mathscr{I}$. Le monôme $y = y^3/y^2$ est le rapport du monôme y^3 à réécrire par le monôme de tête y^2 de la règle avec laquelle on effectue la réduction.

Définition 22 *(formes normales)*
On appelle forme normale *d'un polynôme P par un système de réécriture $\mathscr{S}$ tout polynôme irréductible Q tel que $P \xrightarrow[\mathscr{S}]{*} Q$. Par extension, un polynôme P est on appelé une* forme normale *par $\mathscr{S}$ s'il est irréductible par $\mathscr{S}$.*

Suite de l'exemple. En poussant les réductions, on réécrit y^3 en deux formes normales :

$$(x - x^3) \xleftarrow[\mathscr{S}]{*} y^3 \xrightarrow[\mathscr{S}]{*} x^3.$$

Proposition 16 *Toute combinaison linéaire sur $\mathbb{Q}$ de formes normales est une forme normale.*

Preuve Un polynôme est une forme normale si et seulement s'il est une combinaison linéaire de termes irréductibles (par un certain système de réécriture). Une combinaison linéaire de combinaisons linéaires de termes irréductibles est une combinaison linéaire de termes irréductibles. □

Comme on l'a vu, il n'y a aucune raison que tout polynôme admette une unique forme normale par un système quelconque. Lorsque c'est le cas, le système de réécriture est appelé une « base de Gröbner ».

FIG. 3.1 – Bruno Buchberger est professeur au RISC, à Linz, en Autriche.

3.4 Bases de Gröbner

Bruno Buchberger a inventé les bases de Gröbner en 1965 [4]. Il leur a donné le nom de son directeur de thèse : Wolfgang Gröbner.

Théorème 6 *(théorème et définition — version 1)*
Supposons fixé un ordre admissible sur l'ensemble des termes. Soient $\mathscr{G}$ un ensemble de polynômes de $\mathbb{A}$ et $\mathscr{I}$ l'idéal qu'il engendre. Les conditions suivantes sont équivalentes.

1. *Tout polynôme de $\mathbb{A}$ admet une unique forme normale par $\mathscr{G}$. Deux polynômes équivalents modulo $\mathscr{I}$ ont même forme normale par $\mathscr{G}$.*
2. *Tout polynôme de $\mathscr{I}$ admet zéro pour forme normale par $\mathscr{G}$.*

Un ensemble $\mathscr{G}$ qui satisfait ces conditions est appelé une base de Gröbner *de $\mathscr{I}$.*

Preuve L'implication $\Downarrow$ du haut vers le bas. Tout polynôme $P \in \mathscr{I}$ est équivalent à zéro modulo $\mathscr{I}$. Le polynôme zéro est égal à sa propre forme normale.

L'implication $\Uparrow$ du bas vers le haut. Soient P et Q deux polynômes équivalents modulo $\mathscr{I}$, et $\bar{P}$ et $\bar{Q}$ deux de leurs formes normales (on ne peut pas supposer qu'elles sont uniques !). La différence $\bar{P} - \bar{Q}$ appartient à $\mathscr{I}$. D'après (2) elle se réduit à zéro. Comme elle est déjà une forme normale (proposition 16), elle doit être égale à zéro et donc $\bar{P} = \bar{Q}$. □

Revenons à l'exemple précédent. Le polynôme y^3 admet deux formes normales, ce qui contredit le point (1) du théorème. Le système de réécriture n'est donc pas une base de Gröbner.

Définition 23 *(base de Gröbner réduite)*
Une base de Gröbner $\mathscr{G}$ est dite réduite *si*

1. *tout polynôme $G \in \mathscr{G}$ est irréductible par $\mathscr{G} \setminus \{G\}$ et*
2. *les coefficients numériques des éléments de $\mathscr{G}$ sont « normalisés ».*

Il y a plusieurs façons possibles de normaliser les coefficients numériques : dans le cours, on impose que le coefficient du terme de tête de chaque règle vaille 1 ; dans les logiciels, où on préfère souvent manipuler des coefficients entiers plutôt que rationnels, on impose que tous les coefficients soient entiers, que leur pgcd vaille 1 et que le coefficient du terme de tête soit positif.

Proposition 17 *Toute base de Gröbner peut être transformée en une base de Gröbner équivalente réduite.*

On peut montrer qu'on peut supprimer purement et simplement de $\mathscr{G}$ tout polynôme $G \in \mathscr{G}$ dont le terme de tête est un multiple du terme de tête d'un autre élément de $\mathscr{G}$ (en faisant attention, dans certains cas où deux polynômes ont même terme de tête, à n'en supprimer qu'un des deux).

Proposition 18 *La base de Gröbner réduite d'un idéal $\mathscr{I}$ est unique : elle ne dépend que de l'idéal et de l'ordre admissible choisi.*

3.5 L'algorithme de Buchberger en MAPLE

Tout système d'équations polynomiales $\mathscr{S}$ peut être transformé en une base de Gröbner. L'algorithme de Buchberger prend en entrée un système $\mathscr{S}$ et un ordre admissible. Il produit en sortie une base de Gröbner $\mathscr{G}$. Les ensembles $\mathscr{S}$ et $\mathscr{G}$ sont deux bases du même idéal. Ils ont donc mêmes solutions. En MAPLE, il est implanté dans le paquetage *Groebner* sous le nom *gbasis*. La fonction `gbasis` a deux paramètres : un système de polynômes et un ordre admissible (ici l'ordre lexicographique donné par $y > x$). Elle retourne la base de Gröbner réduite $\mathscr{G}$ de l'idéal engendré par le système. La fonction `leadterm` a deux paramètres : un polynôme et un ordre admissible. Elle retourne le terme de tête du polynôme. La fonction `normalf` a trois paramètres : un polynôme, une base de Gröbner et un ordre admissible. Elle retourne la forme normale du polynôme par la base de Gröbner.

```
    |\^/|     Maple V Release 5 (USTL)
._|\|   |/|_. Copyright (c) 1981-1997 by Waterloo Maple Inc. All rights
 \  MAPLE  /  reserved. Maple and Maple V are registered trademarks of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
with (Groebner);
[fglm, gbasis, gsolve, hilbertdim, hilbertpoly, hilbertseries, inter_reduce,

    is_finite, is_solvable, leadcoeff, leadmon, leadterm, normalf,

    pretend_gbasis, reduce, spoly, termorder, testorder, univpoly]

G := gbasis ([x^2 + y^2 - 1, x - y], plex (y,x));
                                  2
                          G := [2 x  - 1, -x + y]

leadterm (G [1], G, plex (y,x)), leadterm (G [2], G, plex (y,x));
                                    2
```

```
                                    x , y
normalf (x^3 + y, G, plex (y,x));
                                    3/2 x
```

Le même système que précédemment mais pour l'ordre lexicographique donné par $x > y$. On constate que la base de Gröbner a changé.

```
G := gbasis ([x^2 + y^2 - 1, x - y], plex (x,y));
                                    2
                           G := [2 y  - 1, x - y]
```

L'exemple du robot planaire. Les deux segments sont de longueur 1. Les coordonnées de la main dans le repère global sont $(x_0, y_0) = (1/3, 1/2)$. L'ordre admissible est l'ordre lexicographique donné par $c_2 > s_2 > c_1 > s_1$.

```
S := [c1*c2 - s1*s2 + c1 - 1/3, c1*s2 + c2*s1 + s1 - 1/2,
      c1^2 + s1^2 - 1, c2^2 + s2^2 - 1];

                                                                  2     2
   S := [c1 c2 - s1 s2 + c1 - 1/3, c1 s2 + c2 s1 + s1 - 1/2, c1  + s1  - 1,

         2     2
       c2  + s2  - 1]

G := gbasis (S, plex (c2,s2,c1,s1));

                  2
   G := [1872 s1  - 407 - 936 s1, 24 c1 + 36 s1 - 13, 48 s2 + 52 s1 - 13,

       72 c2 + 59]
```

3.6 Décider si un système admet au moins une solution

Le théorème et la proposition qui suivent montrent qu'on peut décider automatiquement si un système d'équations polynomiales n'admet aucune solution complexe : il suffit de tester si le polynôme 1 appartient à une base de Gröbner quelconque de l'idéal engendré par le système.

Théorème 7 *Soient $\mathscr{S}$ un système et $\mathscr{I}$ l'idéal qu'il engendre dans $\mathbb{A}$. Le système n'admet aucune solution dans $\mathbb{C}^n$ si et seulement si $1 \in \mathscr{I}$.*

Ce théorème est un corollaire du théorème des zéros qu'on étudiera plus loin.

Il est évident que si $1 \in \mathscr{I}$ alors le système n'admet aucune solution. C'est l'autre implication qui est difficile à prouver.

On parle bien de solutions complexes et pas réelles : l'équation $x^2 = -1$ n'admet aucune solution réelle mais 1 n'appartient pas à l'idéal qu'elle engendre.

Proposition 19 *Soit $\mathscr{G}$ une base de Gröbner réduite d'un idéal $\mathscr{I}$ pour un quelconque ordre admissible. Alors $1 \in \mathscr{I}$ si et seulement si $\mathscr{G} = \{1\}$.*

Preuve Tout polynôme de $\mathscr{I}$, en particulier 1, est réduit à zéro par $\mathscr{G}$. Il faut donc que $\mathscr{G}$ contienne la règle $1 \longrightarrow 0$. □

3.6.1 Application à la démonstration automatique

Le « truc de Rabinovitch » indique comment transformer une inéquation ($\neq 0$) en équation en introduisant une indéterminée supplémentaire.

Proposition 20 *(truc de Rabinovitch)*
Soient $P \in \mathbb{A} = \mathbb{Q}[x_1, \ldots, x_n]$ un polynôme, x_{n+1} une nouvelle indéterminée et $(\alpha_1, \ldots, \alpha_n) \in \mathbb{C}^n$ un n–uplet de nombres complexes. Définissons le polynôme $\overline{P} = P\,x_{n+1} - 1 \in \mathbb{A}[x_{n+1}]$. Alors

$$P(\alpha_1, \ldots, \alpha_n) \neq 0 \quad \Leftrightarrow \quad \exists \alpha_{n+1} \in \mathbb{C},\ \overline{P}(\alpha_1, \ldots, \alpha_n,\ \alpha_{n+1}) = 0.$$

Dit plus informellement, l'inéquation $P \neq 0$ est équivalente à l'équation $P\,x_{n+1} - 1 = 0$ dans le sens où toute solution de $P \neq 0$ fournit une solution de $P\,x_{n+1} - 1 = 0$ et réciproquement. La nouvelle indéterminée x_{n+1} représente $1/P$.

Le truc de Rabinovitch permet de démontrer automatiquement des théorèmes en effectuant des démonstrations par l'absurde. Montrer par l'absurde une implication $A \Rightarrow B$ consiste à supposer A vrai, B faux et à chercher une contradiction. Dans le cas où les propositions logiques A et B sont des équations $P = 0$ et $Q = 0$, montrer par l'absurde $P = 0 \Rightarrow Q = 0$ consiste à montrer que le système $P = 0,\ Q \neq 0$ est sans solution.

Exemple. On montre automatiquement qu'un polynôme du second degré dont le discriminant est nul n'a qu'une seule racine. Il suffit de montrer que le système suivant est sans solutions dans $\mathbb{C}^5$: les deux premières équations posent que x et y sont deux racines d'un polynôme de degré deux et de coefficients a, b et c ; la troisième que le discriminant est nul ; les deux inéquations que l'équation est bien de degré deux et que les racines sont distinctes.

$$\begin{cases} a\,x^2 + b\,x + c = 0, \\ a\,y^2 + b\,y + c = 0, \\ b^2 - 4\,a\,c = 0, \\ a \neq 0,\ x \neq y. \end{cases}$$

On transforme les inéquations en équations grâce au truc de Rabinovitch. Il suffit donc de montrer que le système suivant est sans solutions dans $\mathbb{C}^6$

$$\begin{cases} a\,x^2 + b\,x + c = 0, \\ a\,y^2 + b\,y + c = 0, \\ b^2 - 4\,a\,c = 0, \\ a\,(x-y)\,z - 1 = 0. \end{cases}$$

Voici la preuve avec MAPLE.

```
S := [a*x^2 + b*x + c, a*y^2 + b*y + c, b^2 - 4*a*c, a*(x-y)*z - 1];
gbasis (S, plex (x,y,a,b,c,z));
                                    [1]
```

3.7 Éliminer des indéterminées

Bien choisir l'ordre admissible permet d'éliminer des indéterminées Dans l'exemple suivant, on cherche à éliminer y. En d'autres termes, on cherche des équations conséquences du système $\mathscr{S}$ qui ne comportent que des x. Pour cela, on choisit l'ordre lexicographique avec $y > x$ (on pose les indéterminées qu'on souhaite éliminer plus grandes que les autres). La base de Gröbner contient le polynôme recherché : $2\,x^2 - 1$.

```
G := gbasis ([x^2 + y^2 - 1, x - y], plex (y,x));
                                      2
                            G := [2 x  - 1, -x + y]
```

La démarche exposée ci–dessus se justifie par le théorème suivant.

Théorème 8 *(théorème d'élimination)*
Soient $\mathscr{G}$ une base de Gröbner d'un idéal $\mathscr{I}$ de $\mathbb{Q}[x_1, \ldots, x_n]$ pour l'ordre lexicographique $x_n > \cdots > x_1$ et $1 \leq \ell \leq n$ un indice. Alors $\mathscr{G} \cap \mathbb{Q}[x_1, \ldots, x_\ell]$ est une base de Gröbner de l'idéal $\mathscr{I} \cap \mathbb{Q}[x_1, \ldots, x_\ell]$ de l'anneau $\mathbb{Q}[x_1, \ldots, x_\ell]$.

Preuve Remarques : l'intersection d'un idéal et d'un anneau est bien un idéal (en particulier, elle n'est jamais vide puisqu'elle contient zéro) ; $\mathscr{I} \cap \mathbb{Q}[x_1, \ldots, x_\ell]$ est un idéal de $\mathbb{Q}[x_1, \ldots, x_\ell]$ mais pas de $\mathbb{Q}[x_1, \ldots, x_n]$.

Pour prouver le théorème, il suffit d'établir que tout $P \in \mathscr{I} \cap \mathbb{Q}[x_1, \ldots, x_\ell]$ admet zéro pour forme normale par $\mathscr{G} \cap \mathbb{Q}[x_1, \ldots, x_\ell]$. Les arguments sont les suivants.

1. Le polynôme P appartient à $\mathscr{I}$ donc $P \xrightarrow[\mathscr{G}]{*} 0$.
2. Considérons la première réduction de la suite $P \xrightarrow[\mathscr{G}]{} P' \xrightarrow[\mathscr{G}]{*} 0$. La tête du polynôme ayant servi à effectuer cette première réduction est un monôme sur l'alphabet $\{x_1, \ldots, x_\ell\}$.
3. Comme l'ordre admissible est un ordre lexicographique où $x_n, \ldots, x_{\ell+1} > x_\ell, \ldots, x_1$ tout polynôme dont la tête est un monôme sur l'alphabet $\{x_1, \ldots, x_\ell\}$ appartient à $\mathbb{Q}[x_1, \ldots, x_\ell]$.
4. Réduire un polynôme de $\mathbb{Q}[x_1, \ldots, x_\ell]$ par un polynôme de $\mathbb{Q}[x_1, \ldots, x_\ell]$ produit encore un polynôme de $\mathbb{Q}[x_1, \ldots, x_\ell]$.

□

On peut généraliser le théorème précédent en remplaçant l'ordre lexicographique par n'importe quel ordre lexicographique par blocs $(x_n, \ldots, x_{\ell+1}) \gg (x_\ell, \ldots, x_1)$.

3.7.1 Application à la démonstration automatique

Le truc de Rabinovitch permet de démontrer par l'absurde des implications $P = 0 \Rightarrow Q = 0$ lorsqu'on connaît à l'avance P et Q. Le théorème d'élimination permet de démontrer automatiquement des théorèmes même dans le cas où on ne connaît pas à l'avance la conclusion.

Exemple. On considère un polynôme du second degré. Quelles propriétés ses racines satisfont–elles lorsque son discriminant est nul ? Mise en équation : on pose que x et y sont deux racines du polynôme, que le discriminant est nul et que le coefficient dominant a est différent de zéro. Choix de l'ordre admissible : comme on cherche des relations entre les racines x et y, on élimine les coefficients et on choisit donc un ordre lexicographique par blocs

$$(a,\, b,\, c) \gg (x,\, y).$$

```
S := [ a*x^2 + b*x + c, a*y^2 + b*y + c, b^2 - 4*a*c, a*z - 1 ]:
G := gbasis (S, lexdeg ([z,a,b,c],[x,y])):
factor (G [1]);
                                        3
                                  (x - y)
```

Le calcul de base de Gröbner montre que le polynôme $(x-y)^3$ appartient à l'idéal engendré par le système. Par conséquent, quel que soit $(a,\, b,\, c,\, x,\, y,\, z) \in \mathbb{C}^6$ solution du système, on a $x = y$. On a montré que les racines sont confondues.

Variante du même exemple. Quelles propriétés satisfont les coefficients d'un polynôme du second degré ayant une racine commune avec sa dérivée ? La mise en équation est évidente. Choix de l'ordre admissible : comme on cherche des relations qui lient les coefficients, on élimine x et on choisit un ordre lexicographique par bloc

$$x \gg (a,\, b,\, c).$$

```
S := [a*x^2 + b*x + c, 2*a*x + b];
G := gbasis (S, lexdeg ([x], [a,b,c]));

                              2
                G := [b  - 4 a c, b x + 2 c, 2 a x + b]
```

Le calcul de base de Gröbner montre que le discriminant $b^2 - 4\,a\,c$ appartient à l'idéal engendré par le système.

3.8 Élimination et équation aux abscisses

On en vient au point principal du chapitre.

Définition 24 *Soient $\mathscr{S}$ un système de $\mathbb{A}$ n'ayant qu'un nombre fini de solutions dans $\mathbb{C}^n$ et x_1 une indéterminée distinguée (qui joue le rôle d'abscisse).*

On appelle équation aux abscisses *des solutions de $\mathscr{S}$ l'équation en l'indéterminée x_1 de plus petit degré qui admet les abscisses de toutes les solutions de $\mathscr{S}$ pour solutions. Il s'agit de*

$$\prod_{\substack{(\alpha_1,\ldots,\alpha_n)\\ \textit{solution de } \mathscr{S}}} (x_1 - \alpha_1).$$

Cette équation n'existe que dans le cas où les solutions du système n'ont qu'un nombre fini d'abscisses distinctes. Elle existe donc pour tout système n'ayant qu'un nombre fini de solutions. « À un problème d'exposant près » cette équation appartient à l'idéal engendré par le système. C'est la conséquence du théorème suivant, dû à Hilbert :

FIG. 3.2 – David Hilbert, 1862–1943.

Théorème 9 *(théorème des zéros)*
Soient $\mathscr{S}$ un système et $\mathscr{I}$ l'idéal qu'il engendre dans $\mathbb{A}$. Un polynôme P s'annule sur toutes les solutions de $\mathscr{S}$ dans $\mathbb{C}^n$ si et seulement s'il existe un exposant $e \in \mathbb{N}$ tel que $P^e \in \mathscr{I}$.

L'implication de droite à gauche est facile. C'est l'autre qui est difficile.

Exemple. Dans l'avant–dernier exemple, on a trouvé que le polynôme $(x-y)^3$ appartenait à l'idéal engendré par le système. Le polynôme $x-y$ s'annule sur toutes les solutions du système. Il n'appartient pas à l'idéal mais une de ses puissances si.

3.9 Décider si un système admet un nombre fini de solutions

Proposition 21 *Soient $\mathscr{S}$ un système de $\mathbb{A}$ et $\mathscr{G}$ une base de Gröbner de l'idéal qu'il engendre pour un ordre admissible quelconque.*

Le système $\mathscr{S}$ admet un nombre fini de solutions dans $\mathbb{C}^n$ si et seulement si, quel que soit $1 \leq k \leq n$, la base $\mathscr{G}$ contient une règle dont le terme de tête est une puissance de x_k.

Preuve L'implication $\Rightarrow$ de gauche à droite. D'après le théorème des zéros, quel que soit k, l'idéal contient une puissance de l'équation aux abscisses (en prenant x_k pour abscisse). Cette équation doit être réduite à zéro par $\mathscr{G}$. Il faut donc que $\mathscr{G}$ contienne une règle dont le terme de tête est une puissance de x_k.

L'implication $\Leftarrow$ de droite à gauche. L'hypothèse est vérifiée en particulier pour l'ordre lexicographique $x_n > \cdots > x_1$. Pour cet ordre, la base de Gröbner contient donc un polynôme ne

dépendant que de l'indéterminée x_1. Par conséquent, dans les solutions du système, x_1 ne prend qu'un nombre fini de valeurs distinctes. Le même raisonnement tenu pour chaque indéterminée x_j ($2 \leq j \leq n$) montre que dans les solutions du système, x_j ne prend qu'un nombre fini de valeurs. Les solutions dans $\mathbb{C}^n$ du système sont donc en nombre fini. □

On parle bien de solutions complexes et pas réelles. Le polynôme $x^2 + y^2$ admet une infinité de solutions complexes mais une seule solution réelle.

3.10 Bases de Gröbner sous forme résolue

Définition 25 *Une base de Gröbner pour l'ordre lexicographique $x_n > \cdots > x_1$ est dite* sous forme résolue *si elle est de la forme*

$$P(x_1) = 0,\ x_2 = F_2(x_1),\ \ldots,\ x_n = F_n(x_1)$$

où P et les F_i sont des polynômes de $\mathbb{Q}[x_1]$.

Si une base de Gröbner est sous forme résolue alors ses solutions sont en nombre fini. La réciproque est fausse. C'est ce que montre l'exemple suivant (quatre solutions mais seulement deux abscisses distinctes).

```
G := gbasis ([x^2 - 1, y^2 - 1], plex (y,x));
```

```
                                   2       2
                          G := [x  - 1, y  - 1]
```

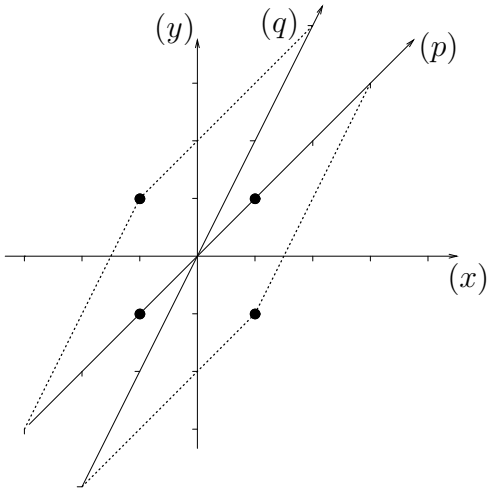
Il existe des techniques pour transformer une base de Gröbner d'un système ayant un nombre fini de solutions en une ou plusieurs bases de Gröbner sous forme résolue [10, 18] et [13, 16, Lex-Triangular]. La méthode la plus simple (mais pas la plus efficace) consiste à tenter un changement de repère « aléatoire » de façon à ce que toutes les solutions aient des abscisses distinctes. Cette méthode ne fonctionne pas si plusieurs solutions sont confondues. Sur l'exemple précédent, cela donne

```
x := p + q:
y := p + 2*q:
H := gbasis ([x^2 - 1, y^2 - 1], plex (q,p));
```

```
                                2         4   3
                  H := [-10 p  + 9 + p , p  - p + 12 q]
```

```
factor (H[1]);
```

```
                     (p - 1) (p + 3) (p - 3) (p + 1)
```



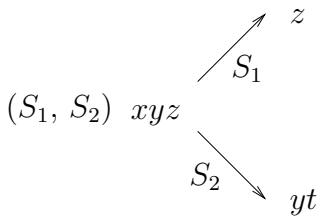
3.11 L'algorithme de Buchberger

Une base de Gröbner $\mathscr{G}$ d'un idéal $\mathscr{I}$ est un système de réécriture qui réécrit à zéro tout élément de $\mathscr{I}$ (et seulement les éléments de $\mathscr{I}$).

Le seul phénomène qui puisse empêcher un système quelconque d'être une base de Gröbner, c'est la présence d'un « conflit » entre deux règles de réécriture. Intuitivement, il y a conflit entre deux règles de réécriture au sujet d'un terme t si appliquer l'une ou l'autre sur t n'est pas indifférent, c'est–à–dire si appliquer l'une ou l'autre donne des formes normales de t différentes. Par exemple, les deux règles suivantes

$$(S_1)\ x\,y \longrightarrow 1, \quad (S_2)\ x\,z \longrightarrow t.$$

sont en conflit au sujet du terme $t_{12} = x\,y\,z$ (c'est le plus petit multiple commun des termes de tête de S_1 et de S_2). Le conflit vient de ce que les deux termes de tête ont une indéterminée commune. On vérifie ci–dessous que, suivant qu'on applique la première ou la deuxième règle, on obtient des formes normales différentes :



Nous avons prouvé que le système de réécriture n'est pas une base de Gröbner.

L'idée appliquée par l'algorithme de Buchberger consiste à « résoudre » le conflit en rajoutant la différence $z - y\,t$ des deux formes normales au système. Ce polynôme appartient[1] à l'idéal $\mathscr{I}$ engendré par les deux règles S_1 et S_2 : on change le système de réécriture mais pas l'idéal. Rajouter

[1]Comme $x\,y\,z \equiv z \mod \mathscr{I}$ et $x\,y\,z \equiv y\,t \mod \mathscr{I}$ on a $z \equiv y\,t \mod \mathscr{I}$ et donc $z - y\,t \in \mathscr{I}$.

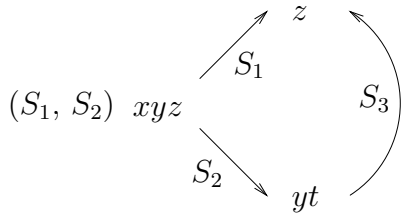
$z - y\,t$ au système, c'est rajouter l'une des règles ci–dessous (ça dépend de l'ordre admissible choisi)

$$z \longrightarrow y\,t \quad \text{ou bien} \quad y\,t \longrightarrow z.$$

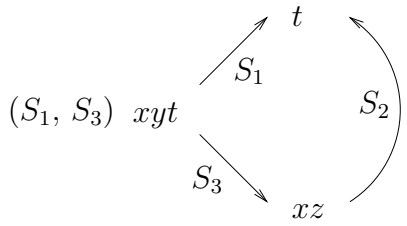
Mettons que l'ordre admissible soit l'ordre lexicographique donné par $x > y > z > t$. Le nouveau système de réécriture est

$$(S_1)\; x\,y \longrightarrow 1, \quad (S_2)\; x\,z \longrightarrow t, \quad (S_3)\; y\,t \longrightarrow z.$$

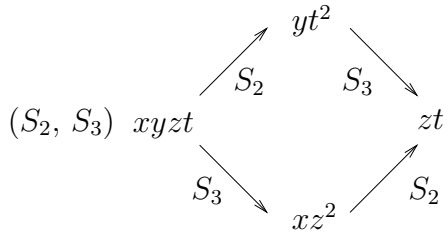
On constate maintenant que t_{12} admet $y\,t$ pour unique forme normale.



Le conflit entre S_1 et S_2 au sujet de t_{12} est donc résolu par l'ajout de la règle S_3 mais l'ajout de S_3 engendre deux nouveaux conflits : il y a un conflit entre S_1 et S_3 au sujet de $t_{13} = x\,y\,t$; il y en a un autre entre S_2 et S_3 au sujet de $t_{23} = x\,z\,y\,t$. On constate que le conflit entre S_1 et S_3 est déjà résolu :



Quant au conflit entre S_2 et S_3, on peut démontrer (sans faire de calcul) qu'il n'existe pas : les deux termes de tête n'ont pas d'indéterminée commune (on dit qu'ils sont disjoints). Appliquer l'une des deux règles n'empêche pas d'appliquer l'autre. Vérification :



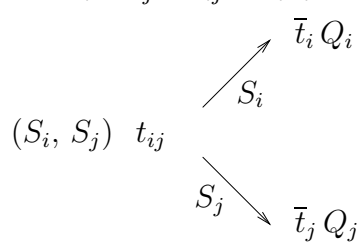
On peut montrer que si pour toute paire de règles d'un système le conflit au sujet du ppcm des têtes de règles est résolu alors le système est une base de Gröbner.

Le système obtenu est donc une base de Gröbner de l'idéal $\mathscr{I}$.

Dans les ouvrages traditionnels consacrés aux bases de Gröbner, les choses sont présentées un peu différemment. Soit à tester si le conflit entre deux règles

$$(S_i)\; t_i \longrightarrow Q_i, \quad (S_j)\; t_j \longrightarrow Q_j$$

est résolu. Notons t_{ij} le ppcm des termes t_i et t_j et $t_{ij} = \overline{t}_i\, t_i = \overline{t}_j\, t_j$. Calculons



Plutôt que tester si $\overline{t}_i\, Q_i$ et $\overline{t}_j\, Q_j$ admettent une même forme normale, on teste si leur différence admet zéro pour forme normale (c'est pareil). La différence est appelée le S–polynôme entre S_i et S_j :

$$S(S_i,\, S_j) \underset{\text{def}}{=} \overline{t}_i\, Q_i - \overline{t}_j\, Q_j.$$

Voici une version en pseudo code de l'algorithme de Buchberger. Il prend en entrée un système de polynômes $\mathscr{S}$ et un ordre admissible (ici sous–entendu). Il retourne une base de Gröbner (pour l'ordre admissible choisi), non nécessairement réduite, de l'idéal engendré par $\mathscr{S}$. On peut facilement l'améliorer pour éviter de traiter les paires de polynômes dont les termes de tête sont disjoints.

```
function Buchberger (𝒮)
begin
  𝒢 := 𝒮
  𝒫 := l'ensemble de toutes les paires {G, G'} ⊂ 𝒢
  while 𝒫 n'est pas vide do
    {G, G'} := un élément de 𝒫
    ôter cette paire de 𝒫
    Ḡ := une forme normale de S(G, G') par 𝒢
    if Ḡ ≠ 0 then
      ajouter à 𝒫 toutes les paires {Ḡ, G} telles que G ∈ 𝒢
      ajouter Ḡ à 𝒢
    fi
  od
end
```

Voici une trace de la fonction Buchberger sur l'exemple. On suppose que l'ordre admissible est l'ordre lexicographique donné par $x > y > z > t$. L'algorithme

	$\mathscr{G}$	$\mathscr{P}$
début du 1er tour	$\{(S_1)\; x\, y \to 1,\; (S_2)\; x\, z \to t\}$	$\{\{S_1,\, S_2\}\}$
début du 2ème tour	$\{(S_1)\; x\, y \to 1,\; (S_2)\; x\, z \to t,\; (S_3)\; \underbrace{y\, t \to z}_{\overline{G}}\}$	$\{\{S_1,\, S_3\},\, \{S_2,\, S_3\}\}$
début du 3ème tour	$\{(S_1)\; x\, y \to 1,\; (S_2)\; x\, z \to t,\; (S_3)\; y\, t \to z\}$	$\{\{S_2,\, S_3\}\}$
fin du 3ème tour	$\{(S_1)\; x\, y \to 1,\; (S_2)\; x\, z \to t,\; (S_3)\; y\, t \to z\}$	$\varnothing$

Quand l'algorithme s'arrête, le système $\mathscr{G}$ satisfait le troisième point du théorème suivant.

Théorème 10 *(théorème et définition — version 2)*

Supposons fixé un ordre admissible sur l'ensemble des termes. Soient $\mathscr{G}$ un ensemble de polynômes de $\mathbb{A}$ et $\mathscr{I}$ l'idéal qu'il engendre. Les conditions suivantes sont équivalentes.

1. *Tout polynôme de $\mathbb{A}$ admet une unique forme normale par $\mathscr{G}$. Deux polynômes équivalents modulo $\mathscr{I}$ ont même forme normale par $\mathscr{G}$.*
2. *Tout polynôme de $\mathscr{I}$ admet zéro pour forme normale par $\mathscr{G}$.*
3. *Tout S–polynôme $S(G,\, G')$ (pour tous $G,\, G' \in \mathscr{G}$) admet zéro pour forme normale par $\mathscr{G}$.*

Un ensemble $\mathscr{G}$ qui satisfait ces conditions est appelé une base de Gröbner *de $\mathscr{I}$.*

La proposition suivante précise le critère que nous avons utilisé sur l'exemple pour éviter de calculer certains S–polynômes.

Proposition 22 *(premier critère de Buchberger pour éviter des calculs inutiles)*

Soient S_1 et S_2 deux règles de réécriture. Si le ppcm des termes de tête des deux règles est égal à leur produit alors le S–polynôme engendré par ces deux règles est réduit à zéro par $\{S_1,\, S_2\}$.

3.11.1 Un système réduit à un unique polynôme est une base de Gröbner

Lorsqu'on lui fournit un système réduit à un unique polynôme, l'algorithme de Buchberger s'arrête immédiatement et retourne ce polynôme.

3.11.2 L'algorithme de Buchberger préserve la rationnalité

On voit que si on fournit à l'algorithme de Buchberger un système à coefficients rationnels, il produit un système à coefficients rationnels.

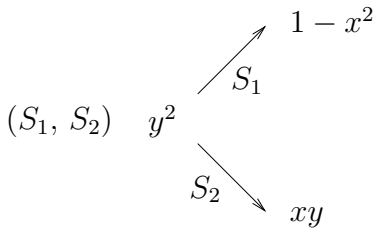
3.11.3 L'intersection d'une droite et d'un cercle

Reprenons l'exemple de l'intersection d'une droite et d'un cercle.

$$x^2 + y^2 - 1 = 0, \quad x - y = 0.$$

Prenons pour ordre admissible l'ordre lexicographique avec $y > x$. Le système de réécriture est

$$(S_1)\; y^2 \longrightarrow 1 - x^2, \quad (S_2)\; y \longrightarrow x.$$



Le S–polynôme est $S(S_1,\, S_2) = 1 - x^2 - x\,y$. Une forme normale du S–polynôme est $1 - 2\,x^2$. On la rajoute au système.

$$(S_1)\; y^2 \longrightarrow 1 - x^2, \quad (S_2)\; y \longrightarrow x, \quad (S_3)\; x^2 \longrightarrow 1/2.$$

Le conflit entre S_1 et S_2 est résolu par l'ajout de la nouvelle règle. Il n'y a pas de conflit entre les deux premières règles et la troisième puisque les termes de têtes sont disjoints. Le système est donc une base de Gröbner.

Le terme de tête de S_1 est divisible par le terme de tête de S_2. La base de Gröbner n'est donc pas réduite. Pour la réduire, il suffit de supprimer S_1 (proposition 17). On obtient finalement :

$$(S_2)\; y \longrightarrow x, \quad (S_3)\; x^2 \longrightarrow 1/2.$$

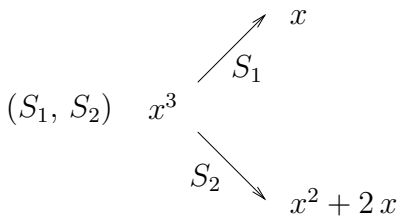
3.11.4 L'algorithme de Buchberger contient l'algorithme d'Euclide

Considérons le système suivant.

$$\begin{cases} x\,(x-1)\,(x+1) &= x^3 - x, \\ (x+1)\,(x-2) &= x^2 - x - 2. \end{cases}$$

Comme il n'y a qu'une seule indéterminée, il n'y a qu'un seul ordre admissible et le système de réécriture est nécessairement

$$(S_1)\; x^3 \longrightarrow x, \quad (S_2)\; x^2 \longrightarrow x + 2.$$



Le S–polynôme est $S(1, 2) = -x^2 - x$. Une forme normale du S–polynôme est $-2\,x - 2$. Elle est égale au reste de la division euclidienne de S_1 par S_2. On la rajoute au système de réécriture

$$(S_1)\; x^3 \longrightarrow x, \quad (S_2)\; x^2 \longrightarrow x + 2, \quad (S_3)\; x \longrightarrow -1.$$

Le conflit entre S_1 et S_2 est résolu par l'ajout de la nouvelle règle. L'algorithme de Buchberger vérifie que le conflit entre S_1 et S_3 est résolu, de même que celui entre S_2 et S_3. Dit autrement, il vérifie que le reste de la division euclidienne de S_1 par S_3 et celui de S_2 par S_3 sont nuls. Par comparaison, l'algorithme d'Euclide est plus adroit : il se contente de vérifier que le reste de la division euclidienne de S_2 par S_3 est nul. On voit sur l'exemple que l'algorithme de Buchberger effectue tous les calculs de reste de l'algorithme d'Euclide plus quelques autres (inutiles). Il contient donc une version maladroite de cet algorithme.

Le système précédent est une base de Gröbner non réduite. Lorsqu'on la réduit, il ne reste que le pgcd des deux polynômes

$$(3)\; x \longrightarrow -1.$$

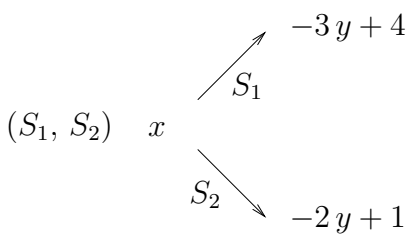
3.11.5 L'algorithme de Buchberger contient le pivot de Gauss

Considérons un système d'équations linéaires.

$$x + 3\,y - 4 = 0, \quad x + 2\,y - 1 = 0, \quad y - z = 0.$$

Adoptons l'ordre lexicographique donné par $x > y > z$. Le système de réécriture obtenu est :

$$(S_1)\; x \longrightarrow -3\,y + 4, \quad (S_2)\; x \longrightarrow -2\,y + 1, \quad (S_3)\; y \longrightarrow z.$$



Le S–polynôme $S(S_1,\, S_2)$ vaut $-y + 3$. Il est égal à l'équation linéaire qu'on aurait obtenue en effectuant une étape du pivot de Gauss entre S_1 et S_2. Une forme normale du S–polynôme est $-z + 3$. On la rajoute au système

$$(S_1)\; x \longrightarrow -3\,y + 4, \quad (S_2)\; x \longrightarrow -2\,y + 1, \quad (S_3)\; y \longrightarrow z, \quad (S_4)\; z \longrightarrow 3.$$

Le conflit entre S_1 et S_2 est résolu par l'ajout de la règle S_4. Il n'y a pas d'autre conflit puisque les termes de tête sont disjoints.

On voit donc que si on lui fournit un système d'équations linéaires à traiter, l'algorithme de Buchberger n'engendre que des équations linéaires. Il contient une variante du pivot de Gauss. Choisir un ordre admissible revient, en termes de pivot de Gauss, à choisir l'ordre des colonnes dans la matrice des coefficients du système.

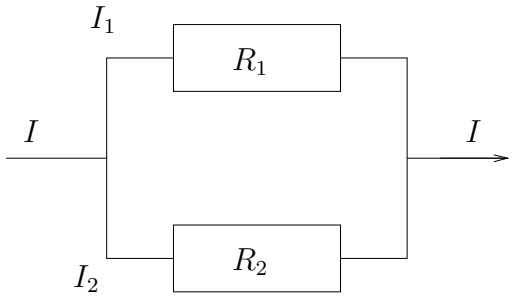
La base de Gröbner réduite du système s'obtient en supprimant soit S_1 soit S_2 mais pas les deux et en réduisant les règles restantes deux à deux. Elle correspond à ce que produirait l'algorithme de Gauss–Jordan.

$$(1)\; x \longrightarrow -5, \quad (3)\; y \longrightarrow 3, \quad (4)\; z \longrightarrow 3.$$

3.11.6 L'algorithme de Buchberger préserve les dimensions

Une analyse rapide de la définition des S–polynômes montre que, si on applique l'algorithme de Buchberger à un système d'équations homogène pour certaines dimensions (un système d'équations « qui ont un sens » d'un point de vue physique) alors l'algorithme de Buchberger ne produit que des équations homogènes. En d'autres termes, l'algorithme de Buchberger ne produit que des équations « qui ont un sens ».

Exemple. On montre une relation bien connue sur la résistance d'un circuit formé de deux résistances en parallèle.



On cherche la relation

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2}.$$

On nomme $\overline{R}$, $\overline{R}_1$ et $\overline{R}_2$ les inverses des résistances (des Ohms^{-1}) et $\overline{I}$ l'inverse de l'intensité (des Ampères^{-1}). Les équations sont homogènes pour certaines dimensions (elles ont un sens physique).

$U = R\,I$	Volts $=$ Ohms Ampères
$U = R_1\,I_1$	
$U = R_2\,I_2$	
$I = I_1 + I_2$	Ampères $=$ Ampères
$I\,\overline{I} = 1$	Ampères Ampères^{-1} $=$ Nombre
$R\,\overline{R} = 1$	Ohms Ohms^{-1} $=$ Nombre
$R_1\,\overline{R}_1 = 1$	
$R_2\,\overline{R}_2 = 1$	

On calcule une base de Gröbner pour l'ordre lexicographique

$$\overline{I} > I > I_1 > I_2 > U > R > R_1 > R_2 > \overline{R} > \overline{R}_1 > \overline{R}_2$$

L'algorithme de Buchberger n'engendre que des relations homogènes pour certaines dimensions (qui ont un sens physique).

$$\begin{aligned} R_1\,\overline{R}_1\,I_1 &\longrightarrow I_1 \\ &\longrightarrow U\overline{R}_1 \end{aligned}$$

Exemple de nouvelle relation calculée par l'algorithme :

$$\begin{aligned} I_1 = U\overline{R}_1 \quad & \text{Ampères} = \text{Volts Ohms}^{-1} \\ & \text{Ampères} = (\text{Ohms Ampères})\,\text{Ohms}^{-1} \end{aligned}$$

Le premier polynôme de la base de Gröbner réduite (calculée avec MAPLE) fournit la relation cherchée.

```
G := [R1bar - Rbar + R2bar, R2 R2bar - 1,

  1 - R1 Rbar + R1 R2bar, -R1 - R2 + R2 R1 Rbar,

  R Rbar - 1, R R2 - R1 R2 + R R1, I2 - R2bar U,

  I1 + R2bar U - Rbar U, I - Rbar U, Ibar U - R]
```

3.12 Exemple

On s'intéresse au système

$$x\,y^2 - x = 0, \quad x^2\,y - x - y = 0.$$

On commence par calculer une base de Gröbner de l'idéal $\mathscr{I}$ engendré par $\mathscr{S}$. On choisit pour ordre admissible l'ordre lexicographique donné par $y > x$. Le système de réécriture initial est

$$\begin{array}{lll} (1) & x\,y^2 & \longrightarrow & x \\ (2) & x^2\,y & \longrightarrow & x + y \end{array}$$

La liste de paires critiques à traiter est

$$\mathscr{P} = [(1,\,2)].$$

On traite la paire critique présente dans $\mathscr{P}$. On calcule le ppcm des termes de tête des deux règles. On le réécrit séparément par chacune d'entre elles. Les polynômes obtenus sont des formes normales.

$$x^2 \underset{1}{\longleftarrow} x^2\,y^2 \underset{2}{\longrightarrow} x\,y + y^2.$$

On ajoute au système la règle obtenue par différence des deux formes normales.

$$\begin{array}{lll} (3) & y^2 & \longrightarrow & x^2 - x\,y \end{array}$$

La liste de paires critiques à traiter est maintenant

$$\mathscr{P} = [(1,\,3),\,(2,\,3)].$$

On traite la première paire critique présente dans $\mathscr{P}$. On calcule le ppcm des termes de tête des deux règles. On le réécrit séparément par chacune d'entre elles. L'un des polynômes obtenus peut encore être réécrit par le système. On pousse les réécritures jusqu'au bout.

$$x \underset{1}{\longleftarrow} x\,y^2 \underset{3}{\longrightarrow} x^3 - x^2\,y \underset{2}{\longrightarrow} x^3 - x - y.$$

On ajoute au système la règle obtenue par différence des deux formes normales.

$$\begin{array}{lll} (4) & y & \longrightarrow & x^3 - 2\,x \end{array}$$

La liste de paires critiques à traiter est maintenant

$$\mathscr{P} = [(2,\,3),\,(1,\,4),\,(2,\,4),\,(3,\,4)].$$

On traite la première paire critique présente dans $\mathscr{P}$. On calcule le ppcm des termes de tête des deux règles. On le réécrit séparément par chacune d'entre elles.

$$x\,y + y^2 \underset{2}{\longleftarrow} x^2\,y^2 \underset{3}{\longrightarrow} x^4 - x^3\,y.$$

Il se passe ici un phénomène intéressant. Aucun des polynômes obtenus n'est une forme normale et il y a plusieurs façons possibles de les réécrire, qui conduisent à des formes normales différentes. Si on s'y prend bien, on peut les réécrire tous les deux en la même forme normale :

$$x^2 \xleftarrow[3]{} x\,y + y^2 \xleftarrow[2]{} x^2\,y^2 \xrightarrow[3]{} x^4 - x^3\,y \xrightarrow[2]{} x^4 - x^2 - x\,y \xrightarrow[4]{} x^2.$$

Si on choisit d'effectuer ces réécritures, on trouve que le conflit entre les règles est résolu (ou encore que le S–polynôme entre les deux règles se réduit à zéro).

Mais on pourrait aussi choisir une autre façon de mener les calculs

$$x^2 \xleftarrow[3]{} x\,y + y^2 \xleftarrow[2]{} x^2\,y^2 \xrightarrow[4]{} x^6 - 3\,x^4$$

qui nous donnerait une nouvelle règle $x^6 \longrightarrow 3\,x^4 - x^2$ (et il y a encore d'autres possibilités).

Suivant le choix effectué, les calculs à suivre vont être très différents. Ils peuvent même être de difficultés très différentes. Cependant, la proposition 18 nous garantit que la base de Gröbner *réduite* obtenue à la fin sera la même dans tous les cas.

Pour des raisons de simplicité, on choisit la première façon de mener les calculs, celle qui n'introduit pas de nouvelle règle de réécriture.

La liste de paires critiques à traiter est maintenant

$$\mathscr{P} = [(1,\,4),\,(2,\,4),\,(3,\,4)].$$

On traite la première paire critique présente dans $\mathscr{P}$. On calcule le ppcm des termes de tête des deux règles. On le réécrit séparément par chacune d'entre elles. On pousse les réductions jusqu'à obtenir des formes normales.

Ici aussi, si on choisit bien la façon de mener les réductions, on trouve que le S–polynôme se réduit à zéro et on n'introduit pas de nouvelle règle (mais il y a d'autres façons de mener les calculs, qui produiraient de nouvelles règles).

$$x \xleftarrow[1]{} x\,y^2 \xrightarrow[4]{} x^4\,y - 2\,x^2\,y \xrightarrow[2]{*} x^3 - x - y \xrightarrow[4]{} x.$$

La liste de paires critiques à traiter est maintenant

$$\mathscr{P} = [(2,\,4),\,(3,\,4)].$$

On traite la première paire critique présente dans $\mathscr{P}$. On calcule le ppcm des termes de tête des deux règles. On le réécrit séparément par chacune d'entre elles. On pousse les réductions jusqu'à obtenir des formes normales.

$$x^3 - x \xleftarrow[4]{} x + y \xleftarrow[2]{} x^2\,y \xrightarrow[4]{} x^5 - 2\,x^3.$$

On ajoute au système la règle obtenue par différence des deux formes normales.

$$(5)\quad x^5 \longrightarrow 3\,x^3 - x$$

La liste de paires critiques à traiter est maintenant

$$\mathscr{P} = [(3,\,4),\,(1,\,5),\,(2,\,5)].$$

On remarque qu'on n'a pas inséré dans $\mathscr{P}$ les paires critiques $(3,\,5)$ et $(4,\,5)$ qui sont superflues d'apès la proposition 22.

On laisse au lecteur le soin de vérifier que les S–polynômes engendrés par les trois paires présentes dans $\mathscr{P}$ se réduisent à zéro (quelle que soit la façon dont on mène les calculs d'ailleurs puisqu'on a une base de Gröbner).

Voici donc une base de Gröbner non réduite de l'idéal $\mathscr{I}$.

$$x\,y^2 \longrightarrow x, \quad x^2\,y \longrightarrow x + y, \quad y^2 \longrightarrow x^2 - x\,y, \quad y \longrightarrow x^3 - 2\,x, \quad x^5 \longrightarrow 3\,x^3 - x.$$

Et voici la base de Gröbner réduite :

$$y \longrightarrow x^3 - 2\,x, \quad x^5 \longrightarrow 3\,x^3 - x.$$

3.13 Arrêt de l'algorithme de Buchberger

Proposition 23 *L'algorithme de Buchberger produit une base de Gröbner en un nombre fini d'étapes quels que soient le système et l'ordre admissible en entrée.*

La preuve de cette proposition n'est pas très facile. Elle est fondée sur ce qu'on appelle le « lemme de Dickson » [8].

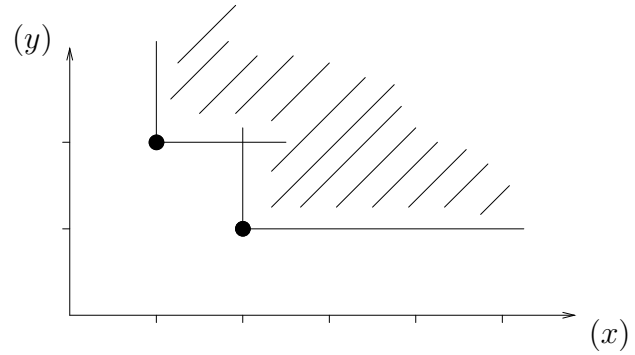
FIG. 3.3 – Leonard Eugene Dickson, 1874–1954.

On se contente de l'illustrer graphiquement sur l'exemple précédent.

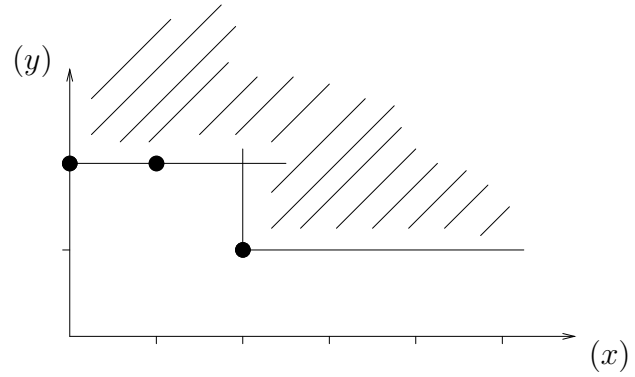
On dessine un diagramme à chaque fois que l'algorithme de Buchberger ajoute une nouvelle règle de réécriture à la base en construction. Sur ces diagrammes on représente l'ensemble des termes. Il y a autant d'axes qu'il y a d'indéterminées (deux axes ici). On représente les termes de

tête des règles par des disques noirs et on hachure l'ensemble des termes qui peuvent être réécrits par le système (tous les multiples des termes de tête).

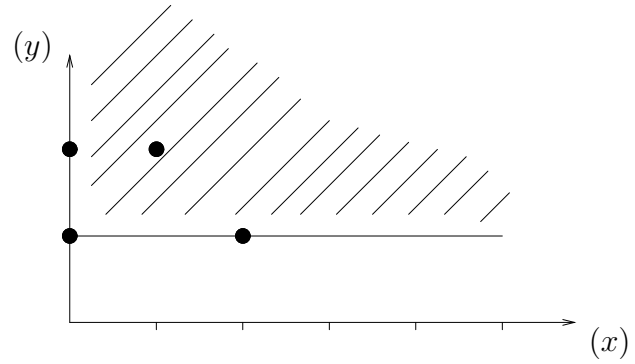
Initialement, les termes de tête sont $x\,y^2$ et $x^2\,y$ et le diagramme



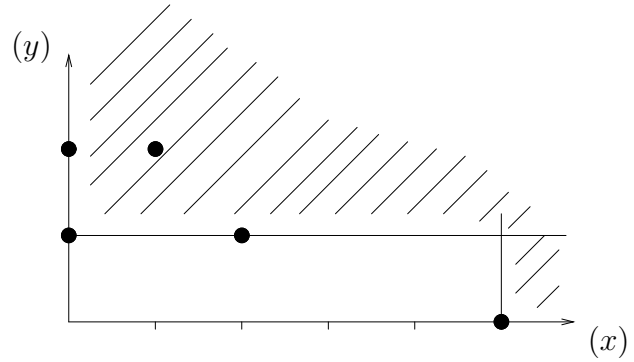
Ensuite l'algorithme insère une règle de terme de tête y^2. Le diagramme devient



Puis il insère une règle de terme de tête y. Le diagramme devient



Enfin, il insère une règle de terme de tête x^5. Le diagramme devient



À chaque fois qu'une nouvelle règle est introduite, l'ensemble des termes irréductibles (ceux qui ne sont pas hachurés) se rétrécit. C'est normal, puisque chaque nouvelle règle est une forme normale (c'est–à–dire une combinaison linéaire de termes figurant dans la partie non hachurée).

Lorsqu'il n'y a que deux indéterminées, on « sent bien » qu'on ne peut pas indéfiniment placer ainsi des disques noirs dans les parties blanches et hachurer le quart de plan supérieur droit.

Ce sentiment peut s'écrire de façon rigoureuse et se généraliser à un nombre quelconque d'indéterminées.

Chapitre 4

Pgcd de polynômes et calcul modulaire

Dans cette section, on développe certaines des notions abordées au début du chapitre 2. On en déduit un algorithme efficace pour calculer le pgcd de deux polynômes en une indéterminée et à coefficients rationnels. Les méthodes utilisées (calcul modulaire) se rencontrent en cryptographie.

4.1 Le problème

Si F et G sont deux polynômes de $\mathbb{Q}[x]$ avec des petits coefficients alors leur pgcd P a généralement des coefficients petits, lui aussi. Par contre la taille des coefficients des restes intermédiaires calculés par l'algorithme d'Euclide croît tellement à chaque itération que cet algorithme en devient impraticable. Voici un exemple où, pour simplifier, on a normalisé à 1 le coefficient dominant de chaque reste.

$$\begin{aligned}
R_0 &= (3\,x-2)\,(x^8+x^6-3\,x^4-3\,x^3-3\,x^2+2\,x-5)\\
R_1 &= (3\,x-2)\,(3\,x^6+5\,x^4-4\,x^2-9\,x+21)\\
R_2 &= x^5-\frac{2}{3}\,x^4+\frac{98}{5}\,x^3-\frac{196}{15}\,x^2+\frac{3}{5}\,x-\frac{2}{5}\\
R_3 &= x^3-\frac{1967}{2913}\,x^2+\frac{494}{8739}\,x-\frac{296}{8739}\\
R_4 &= x^2-\frac{440390801}{145990173}\,x+\frac{228709346}{145990173}\\
R_5 &= x-\frac{2}{3}.
\end{aligned}$$

4.2 Calcul modulaire

L'idée consiste à exécuter l'algorithme d'Euclide non plus dans l'anneau $\mathbb{Q}[x]$ mais dans l'anneau $(\mathbb{Z}/n\mathbb{Z})[x]$ où n est en entier naturel, souvent premier. Les éléments de $\mathbb{Z}/n\mathbb{Z}$ sont les entiers pris « modulo n » (d'où l'expression calcul modulaire) qui ne peuvent donc pas sortir de l'intervalle $[0,\,n-1]$. Toute la difficulté consiste bien sûr à retrouver *in fine* le résultat recherché, c'est–à–dire le pgcd à coefficients rationnels.

4.2.1 Division euclidienne, pgcd dans $\mathbb{Z}$

On rappelle, pour les entiers, quelques notions déjà abordées pour les polynômes.

Proposition 24 *(division euclidienne)*
Soient f et $g \neq 0$ deux entiers naturels. Il existe un unique couple (q, r) d'entiers naturels tels que $f = g\,q + r$ et $r < g$. Les entiers q et r sont appelés le quotient et le reste de la division euclidienne de f par g.

La proposition précédente se généralise au cas d'entiers signés. Dans ce cas, une convention possible consiste à imposer que le couple (q, r) satisfasse : $f = g\,q + r$, $|r| < |g|$ et $f\,r \geq 0$. C'est ce que font les fonctions MAPLE *iquo* et *irem*, qui implantent le quotient et le reste de la division euclidienne de deux entiers. Comme dans le cas des polynômes, un entier f divise un entier g si et seulement si le reste de la division euclidienne de f par g est nul.

Définition 26 *(pgcd)*
Soient f et g deux entiers. Un entier p est un plus grand commun diviseur *(ou* pgcd*) de f et de g s'il satisfait :*

1. *$p \mid f$ et $p \mid g$ (p est un diviseur commun),*
2. *s'il existe p' tel que $p' \mid f$ et $p' \mid g$ alors $p' \mid p$ (p est le plus grand).*

La définition précédente s'applique aussi bien pour les entiers que pour les polynômes. Plus généralement, elle permet de définir le pgcd de deux éléments de n'importe quel anneau où cette notion est bien définie (anneaux « factoriels »). Comme dans le cas des polynômes, le pgcd est défini au produit près par un élément inversible de $\mathbb{Z}$, c'est–à–dire 1 et -1. On dispose pour les entiers d'un algorithme d'Euclide et d'un algorithme d'Euclide étendu. On ne rappelle pas les pseudo–codes qui sont identiques à ceux fournis pour les polynômes. L'algorithme d'Euclide étendu permet le calcul des identités de Bézout. La fonction MAPLE correspondante s'appelle *igcdex*.

```
igcdex (21, 49, 'u', 'v');
                                    7

u, v;
                                 -2, 1

21*u + 49*v;
                                    7
```

4.2.2 Anneau $\mathbb{Z}/n\mathbb{Z}$

Définition 27 *(congruence)*
Soient f, g et $n \neq 0$ trois entiers. On dit que f est congru *à g modulo n (ou encore* équivalent *à g modulo n) si $n \mid (f - g)$. On le note $f \equiv g \mod n$.*

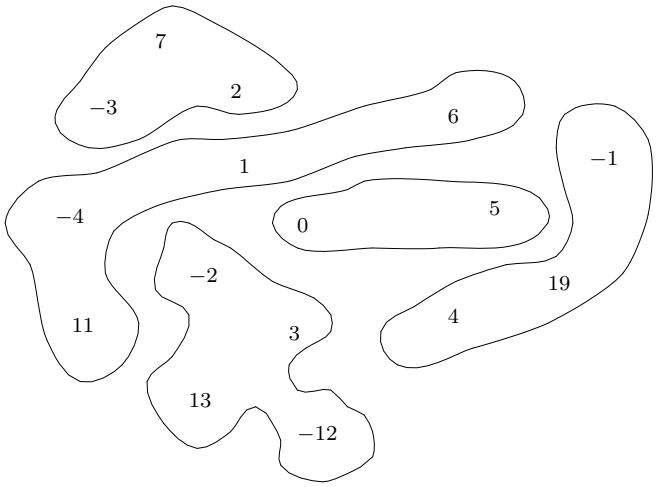


FIG. 4.1 – $\mathbb{Z}/5\mathbb{Z}$ (détail)

La relation de congruence modulo n est une relation d'équivalence. On note $\mathbb{Z}/n\mathbb{Z}$ l'ensemble des classes d'équivalence de $\mathbb{Z}$ pour cette relation. L'ensemble $\mathbb{Z}/n\mathbb{Z}$ comporte exactement n éléments (voir par exemple la figure 4.1).

Proposition 25 *Si* $f \equiv f' \mod n$ *et* $g \equiv g' \mod n$ *alors*

$$f + g \equiv f' + g' \mod n, \quad f \times g \equiv f' \times g' \mod n.$$

Cette proposition a pour conséquence le principe informel suivant : *dans une somme ou un produit (modulo n), on peut toujours remplacer un nombre f par un nombre f' tel que $f \equiv f' \mod n$.* On peut vérifier dans $\mathbb{Z}/5\mathbb{Z}$ par exemple que

$$\begin{array}{ccc} 1 & = & 6 \\ + & & + \\ 3 & = & 13 \\ \| & & \| \\ 4 & = & 19 \end{array} \qquad \begin{array}{ccc} 1 & = & 6 \\ \times & & \times \\ 3 & = & -2 \\ \| & & \| \\ 3 & = & -12 \end{array}$$

Autre conséquence de la proposition 25 : il est possible de définir la « somme » et le « produit » de deux classes d'équivalence modulo n par (la barre signifie « classe d'équivalence ») :

$$\begin{aligned} \overline{f} + \overline{g} &= \overline{f+g}, \\ \overline{f} \times \overline{g} &= \overline{f \times g}. \end{aligned}$$

La proposition assure que la définition est sensée, c'est–à–dire que la classe de la somme (ou du produit) de deux éléments ne dépend que des classes et pas des éléments f ou g choisis. L'ensemble $\mathbb{Z}/n\mathbb{Z}$ est alors muni d'une structure d'anneau (c'est le *quotient* de l'anneau $\mathbb{Z}$ par l'un de ses idéaux $n\mathbb{Z}$). L'élément zéro de $\mathbb{Z}/n\mathbb{Z}$ est l'ensemble des multiples de n. L'élément 1 est

l'ensemble des multiples de n plus 1. En pratique, on représente les classes d'équivalences par un élément distingué, appelé *représentant canonique* de la classe. On prend souvent des représentants positifs : $\mathbb{Z}/5\mathbb{Z} = \{0,\, 1,\, 2,\, 3,\, 4\}$. On utilise aussi parfois des représentants signés : $\mathbb{Z}/5\mathbb{Z} = \{0,\, 1,\, 2, -2,\, -1\}$. En MAPLE, les fonctions *modp* et *mods* permettent d'obtenir l'image modulo n d'un entier dans l'une ou l'autre de ces représentations.

4.2.3 Test d'inversibilité et calcul de l'inverse

Pour additionner, soustraire et multiplier dans $\mathbb{Z}/n\mathbb{Z}$, il suffit d'effectuer l'opération dans $\mathbb{Z}$ sur les représentants des classes puis de prendre le représentant canonique de la classe du résultat (avec des représentants positifs, il suffit de prendre le reste de la division du résultat par n). L'algorithme d'Euclide étendu, qui calcule des identités de Bézout, fournit un moyen d'effectuer des divisions. En fait, il fait deux choses : il permet de tester si un élément de $\mathbb{Z}/n\mathbb{Z}$ est inversible et, si c'est le cas, il fournit l'inverse.

Proposition 26 *(test d'inversibilité et calcul de l'inverse)*
Soit $u\, f + n\, v = p$ une identité de Bézout entre deux entiers f et $n \neq 0$. Le pgcd p vaut 1 si et seulement si f est inversible dans $\mathbb{Z}/n\mathbb{Z}$. Si le pgcd p vaut 1 alors $u = 1/f$ dans $\mathbb{Z}/n\mathbb{Z}$.

Proposition 27 *L'anneau $\mathbb{Z}/n\mathbb{Z}$ est un corps si et seulement si n est un nombre premier.*

4.2.4 Le théorème chinois

Le théorème chinois (on dit aussi « théorème des restes chinois ») résout le problème suivant : connaissant l'image f_1 d'un entier f modulo un entier n_1 et l'image f_2 du même entier f modulo un entier n_2, est–il possible de calculer l'image f_{12} de f modulo le produit $n_1 \times n_2$? La réponse est oui, sous réserve que les entiers n_1 et n_2 soient premiers entre eux.

La partie « algorithmique » du théorème repose sur l'algorithme d'Euclide étendu. On suppose n_1 et n_2 premiers entre eux. Il existe donc une identité de Bézout :

$$u\, n_1 + v\, n_2 = 1.$$

L'image f_{12} de f modulo $n_1 \times n_2$ s'obtient par la formule suivante, qui fait intervenir f_1 et f_2 mais pas f :

$$g_{12} = v\, n_2 \times f_1 + u\, n_1 \times f_2.$$

Voici une formulation classique du théorème chinois :

Proposition 28 *(théorème chinois)*
Si n_1 et n_2 sont deux entiers premiers entre eux alors, pour tout élément $(f_1,\, f_2)$ du produit cartésien $\mathbb{Z}/n_1\mathbb{Z} \times \mathbb{Z}/n_2\mathbb{Z}$, il existe un unique élément f_{12} de l'anneau $\mathbb{Z}/(n_1 \times n_2)\mathbb{Z}$ équivalent à f_1 modulo n_1 et à f_2 modulo n_2.

L'exemple suivant montre qu'au couple $(2,\ 4)$ du produit cartésien $\mathbb{Z}/8\mathbb{Z} \times \mathbb{Z}/21\mathbb{Z}$ correspond l'élément 130 de l'anneau $\mathbb{Z}/168\mathbb{Z}$.

```
f1 := 2:
f2 := 4:
n1 := 8:
n2 := 21:
igcdex (n1, n2, 'u', 'v');
                                        1

f12 := v*n2*f1 + u*n1*f2 mod n1*n2;
                                   f12 := 130

f12 mod n1, f12 mod n2;
                                      2, 4
```

L'ensemble $\mathbb{Z}/(n_1 \times n_2)\mathbb{Z}$ est un anneau. On peut munir aussi le produit cartésien $\mathbb{Z}/n_1\mathbb{Z} \times \mathbb{Z}/n_2\mathbb{Z}$ d'une structure d'anneau (appelé *anneau produit*) en posant que l'addition et le produit de deux couples se font composante par composante. L'élément zéro de l'anneau produit est le couple $(0,\ 0)$. L'élément 1 est le couple $(1,\ 1)$. Avec cette convention, on peut montrer que l'application qui envoie les éléments de l'anneau quotient $\mathbb{Z}/(n_1 \times n_2)\mathbb{Z}$ sur les couples de l'anneau produit est un homomorphisme (ou plus simplement un « morphisme ») d'anneaux : l'image de la somme (ou du produit) de deux éléments de l'anneau quotient est égale à la somme (ou au produit) des images des éléments dans l'anneau produit. Ce morphisme établit une bijection entre les deux anneaux. Les anneaux $\mathbb{Z}/(n_1 \times n_2)\mathbb{Z}$ et $\mathbb{Z}/n_1\mathbb{Z} \times \mathbb{Z}/n_2\mathbb{Z}$ sont donc isomorphes.

4.2.5 Conversions entre rationnels et nombres modulaires

Soient f/g un nombre rationnel et n un entier tel que $g \wedge n = 1$. L'inverse g' de g modulo n existe et on peut définir naturellement l'image de f/g modulo n par :

$$\frac{f}{g} \mod n = f \times g' \mod n.$$

L'ensemble $\mathbb{Q}_n$ des rationnels f/g tels que $g \wedge n = 1$ forme un anneau : la somme et le produit de deux éléments de $\mathbb{Q}_n$ est encore un élément de $\mathbb{Q}_n$. L'application « modulo n » qui envoie les fractions de l'anneau $\mathbb{Q}_n$ dans l'anneau $\mathbb{Z}/n\mathbb{Z}$ est un morphisme d'anneaux : l'image modulo n de la somme (ou du produit) de deux fractions de $\mathbb{Q}_n$ est égale à la somme (ou au produit) des images des deux fractions. Par exemple, dans $\mathbb{Z}/11\mathbb{Z}$,

$$\begin{array}{ccc} 1/2 & = & -5 \\ + & & + \\ 2/3 & = & -3 \\ \| & & \| \\ 7/6 & = & 3 \end{array} \qquad \begin{array}{ccc} 1/2 & = & -5 \\ \times & & \times \\ 2/3 & = & -3 \\ \| & & \| \\ 2/6 & = & 4 \end{array}$$

Peut–on calculer le morphisme inverse, c'est–à–dire l'ensemble des antécédents d'un élément de $\mathbb{Z}/n\mathbb{Z}$ pour cette application ? Cet ensemble comporte une infinité de fractions. Une variante de l'algorithme d'Euclide étendu permet (parfois) de calculer l'une d'elles.

Proposition 29 *Si a est un élément de $\mathbb{Z}/n\mathbb{Z}$ alors il existe au plus un nombre rationnel f/g de $\mathbb{Q}_n$ tel que $f/g \equiv a \mod n$ et $|f|, |g| < \sqrt{n/2}$.*

L'algorithme d'Euclide étendu, appliqué à un élément a de $\mathbb{Z}/n\mathbb{Z}$ et à n, retourne ce rationnel s'il existe. Il suffit de changer la fonction en arrêtant les calculs dès que $v_3 = 0$ ou que $|u_3|, |u_1| < \sqrt{n/2}$. Si on arrête parce que $v_3 = 0$, c'est que le rationnel recherché n'existe pas. Si on s'arrête parce que $|u_3|, |u_1| < \sqrt{n/2}$, il reste à vérifier si $u_1 \wedge n = 1$. Si c'est le cas, on retourne la fraction $f/g = u_3/u_1$ sinon, c'est que la conversion est impossible. Cet algorithme est dû à Paul Shyh–Horng Wang [23, 20]. Il est assez peu connu (il n'est pas implanté en MAPLE). Sa preuve n'est pas facile : toute la difficulté consiste à montrer que si l'algorithme ne retourne pas de rationnel, alors ce rationnel n'existe pas. Voici une implantation en MAPLE de l'algorithme de Wang.

```
wang := proc (a, n)
     local u, v, t, q;
     u := [1, 0, a];
     v := [0, 1, n];
     while v[3] <> 0 and (u[3]^2 >= n/2 or u[1]^2 >= n/2) do
         q := iquo (u[3], v[3]);
         t := v;
         v := u - q*v;
         u := t
     od;
     if v[3] = 0 then
         error ("rational does not exist")
     elif igcd (u[1], n) <> 1 then
         error ("inversion of a zero divisor")
     else
         u[3]/u[1]
     fi
end:
```

Pour additionner les deux fractions ci–dessous, on peut soit le faire directement dans $\mathbb{Q}$ (ou dans $\mathbb{Q}_n$), soit prendre leur image modulo un entier n suffisamment grand, additionner les images et les « remonter » en rationnels par l'algorithme de Wang. On obtient le même résultat.

```
n := 65521:
a := 2/7:
b := -8/13:
a+b;
                                    -30
                                    ---
                                    91
wang (a+b mod n, n);
                                    -30
                                    ---
                                    91
```

4.3 Application du calcul modulaire au pgcd de deux polynômes

4.3.1 Une première méthode, présentée sur un exemple

On illustre l'idée sur l'exemple introductif. On choisit un nombre premier n et on applique l'algorithme d'Euclide aux deux polynômes R_0 et R_1 dans $(\mathbb{Z}/n\mathbb{Z})[x]$. Les coefficients des polynômes sont pris modulo n : ils ne peuvent pas dépasser $n-1$. Les divisions euclidiennes effectuées par l'algorithme d'Euclide ne posent pas de difficulté puisque, n étant premier, $\mathbb{Z}/n\mathbb{Z}$ est un corps : il suffit d'utiliser l'algorithme d'Euclide étendu (dans $\mathbb{Z}$) pour inverser le coefficient dominant du polynôme par lequel on divise. Voici la suite des restes calculés pour $n = 101$ (des représentants positifs sont pris à partir de R_2). On a normalisé à 1 les coefficients dominants :

$$\begin{aligned}
R_0 &= (3\,x-2)\,(x^8+x^6-3\,x^4-3\,x^3-3\,x^2+2\,x-5)\\
R_1 &= (3\,x-2)\,(3\,x^6+5\,x^4-4\,x^2-9\,x+21)\\
R_2 &= x^5+33\,x^4+60\,x^3+61\,x^2+41\,x+40\\
R_3 &= x^3+3\,x^2+36\,x+23\\
R_4 &= x^2+39\,x+97,\\
R_5 &= x+33.
\end{aligned}$$

On obtient le pgcd de R_0 et de R_1 dans $\mathbb{Q}[x]$ en appliquant l'algorithme de Wang sur les coefficients de R_5 :

```
n := 101:
wang (1, n)*x + wang (33, n);
                                    x - 2/3
```

4.3.2 Ce que cache l'exemple

L'exemple précédent fonctionne parfaitement pour deux raisons :

1. le pgcd de R_0 et de R_1 calculé modulo 101 est l'image modulo 101 du pgcd dans $\mathbb{Q}[x]$;
2. le nombre premier 101 est suffisamment grand par rapport aux coefficients 1 et $2/3$ du pgcd dans $\mathbb{Q}[x]$.

Aucun de ces deux points n'est assuré dans le cas général. Le premier est appelé le « problème du degré » ; le second, le « problème du coefficient initial ».

4.3.3 Nombres premiers malchanceux

C'est parce que l'application « modulo n » est un morphisme de l'anneau $\mathbb{Q}_n$ dans $\mathbb{Z}/n\mathbb{Z}$ et que les coefficients dominants des restes calculés par l'algorithme d'Euclide sont tous non nuls modulo 101 que le pgcd de R_0 et de R_1 calculé modulo 101 est l'image modulo 101 du pgcd dans $\mathbb{Q}[x]$. Ainsi, sur l'exemple, les degrés des restes ne dégénèrent pas modulo 101. Mais pour certains nombres premiers n, appelés « premiers malchanceux », il peut y avoir dégénerescence. Dans ce cas, le pgcd calculé modulo n est de degré différent du pgcd calculé dans $\mathbb{Q}[x]$. C'est ce qu'illustrent les exemples ci–dessous :

```
Gcd (R[0], R[1]) mod 3;
                                    1

Gcd (R[0], R[1]) mod 5;
                                   2
                                  x  + 3 x + 2

Gcd (R[0], R[1]) mod 17;
                                3       2
                               x  + 15 x  + x + 10
```

Remarquer le « G » majuscule de la fonction « Gcd », qui fait que le pgcd est calculé modulo n. Avec un « g » minuscule, on obtiendrait l'image modulo n du pgcd dans $\mathbb{Q}[x]$.

4.3.4 Pgcd dans $\mathbb{Z}[x]$

L'anneau $\mathbb{Z}[x]$ des polynômes à coefficients entiers est un anneau factoriel. La notion de pgcd (définition 26) y est bien définie. Si F et G sont deux polynômes à coefficients entiers, il est possible de les considérer soit comme des éléments de $\mathbb{Z}[x]$ soit comme des éléments de $\mathbb{Q}[x]$. On peut montrer que les pgcd de F et G dans $\mathbb{Z}[x]$ sont des pgcd de F et G dans $\mathbb{Q}[x]$. La réciproque est fausse. Sans être bien difficile, l'affirmation ci–dessus n'est pas absolument évidente. Elle repose sur la proposition suivante.

Proposition 30 *Tout polynôme P à coefficients entiers, qui n'est pas irréductible dans $\mathbb{Q}[x]$, se factorise en un produit de deux polynômes à coefficients entiers F et G tels que $0 < \deg F, \deg G < \deg P$.*

On voit donc que pour calculer le pgcd de deux polynômes F et G de $\mathbb{Q}[x]$, il est possible de chasser les dénominateurs de leurs coefficients et de calculer le pgcd dans $\mathbb{Z}[x]$ des deux polynômes ainsi obtenus. On évite ainsi de considérer le problème des fractions dont les dénominateurs ne seraient pas inversibles modulo n.

La propriété élémentaire suivante est la clef de nombreux problèmes.

Proposition 31 *Soient F et G deux polynômes de $\mathbb{Z}[x]$ et P leur pgcd. Le coefficient dominant de P divise le coefficient dominant de F et celui de G.*

4.3.5 Le problème du degré

Soient F et G deux polynômes à coefficients entiers et P leur pgcd dans $\mathbb{Z}[x]$. Soient n un nombre premier et P_n le pgcd de F et G calculé dans $(\mathbb{Z}/n\mathbb{Z})[x]$. On suppose pour simplifier que le coefficient dominant de P_n est égal à 1.

Pour résoudre le problème du degré, l'idée consiste à choisir n de telle sorte que le degré de P_n soit supérieur ou égal au degré de P. Comme P est le polynôme de plus grand degré qui divise à la fois F et G, on pourra détecter les nombres premiers malchanceux en testant si F et G sont divisibles ou pas par le polynôme de $\mathbb{Z}[x]$ déduit de P_n.

La proposition suivante est une conséquence de la proposition 31.

Proposition 32 *Si n ne divise pas le coefficient dominant de F ou celui de G alors le degré de P_n est supérieur ou égal au degré de P.*

Proposition 33 *Si n ne divise pas le coefficient dominant de F ou celui de G et si P_n et P ont même degré alors il existe $a \in \mathbb{Z}/n\mathbb{Z}$ tel que $P \mod n = a\, P_n$.*

Proposition 34 *Les nombres premiers tels que $\deg P_n \neq \deg P$ (nombres premiers « malchanceux ») sont en nombre fini.*

4.3.6 Le problème du coefficient dominant

Soient F et G deux polynômes à coefficients entiers et P leur pgcd dans $\mathbb{Z}[x]$. Soient n un nombre premier et P_n le pgcd de F et G calculé dans $(\mathbb{Z}/n\mathbb{Z})[x]$. On suppose pour simplifier que le coefficient dominant de P_n est égal à 1.

Pour pouvoir appliquer l'algorithme de Wang sur P_n, il suffit de disposer d'une borne sur les coefficients de P (proposition 29). En voici une, due pour l'essentiel à Edmund Landau [12] et (indépendemment) à Maurice Mignotte [14]. En toute rigueur, ce qu'on appelle la borne de Landau et Mignotte est un résultat intermédiaire qui permet de prouver la proposition 35. C'est un peu abusivement qu'on qualifie la proposition 35 de « borne de Landau et Mignotte » dans ce support de cours. Voir [15, pages 158–169].

Définition 28 *On appelle* hauteur *d'un polynôme P à coefficients entiers le maximum des valeurs absolues de ses coefficients. On la note $H(P)$.*

En MAPLE, la fonction *maxnorm* retourne la hauteur d'un polynôme.

Proposition 35 *(borne de Landau et Mignotte)*
Soient F et G deux polynômes de $\mathbb{Z}[x]$. Si F divise G alors $H(F) < 2^{\deg G}\, H(G)$.

On remarque que la hauteur de P peut fort bien être supérieure à celles de F et G (exemple de $F = x^3 + x^2 - x - 1$ et $G = x^4 + x^3 + x + 1$ qui ont pour pgcd $P = x^2 + 2\,x + 1$).

En fait, la borne de Landau et Mignotte permet d'éviter totalement l'algorithme de Wang. La clef vient de la proposition 31 : si on multiplie P_n par le pgcd p des coefficients dominants de F et de G, on obtient un polynôme $p\,P_n$ qui est l'image dans $(\mathbb{Z}/n\mathbb{Z})[x]$ de P à un facteur *entier* (donc non fractionnaire) près. Pour effectuer la remontée dans $\mathbb{Z}[x]$, il suffit de prendre des représentants *signés* pour les classes d'équivalence de $\mathbb{Z}/n\mathbb{Z}$ ainsi qu'un nombre premier n tel que

$$n > 2 \times p \times \min\left(2^{\deg F}\, H(F),\, 2^{\deg G}\, H(G)\right).$$

Reprenons l'exemple de la section 4.3.1. D'après la borne de Landau et Mignotte, il suffit de prendre un nombre premier supérieur ou égal à 58369 or $n = 101$ suffit. La borne de Landau et Mignotte est en fait souvent pessimiste.

```
p := igcdex (lcoeff (R[0]), lcoeff (R[1]));
                                    p := 3

b0 := 2^degree (R [0])*maxnorm (R [0]);
                                  b0 := 9728

b1 := 2^degree (R [1])*maxnorm (R [1]);
                                  b1 := 10368

nextprime (2 * p * min (b0, b1));
                                    58369
```

On effectue la remontée de $P_n = x + 33$ dans $\mathbb{Z}[x]$ de la façon suivante. Sur l'exemple, on a obtenu P. En général, on n'obtient seulement qu'un multiple du pgcd.

```
> 'mod' := mods;
                                  mod := mods

> p * (x + 33) mod n;
                                    3 x - 2
```

4.3.7 Un algorithme pour la première méthode

Il reste quelques détails mineurs à régler dus au fait que F et G peuvent avoir un facteur commun entier.

Définition 29 *Soit F un polynôme de $\mathbb{Z}[x]$. On appelle* contenu *de F le pgcd de ses coefficients. La* partie primitive *de F s'obtient en divisant F par son contenu. Un polynôme dont le contenu vaut 1 est dit* primitif.

Les fonctions MAPLE *icontent* et *iprimpart* permettent d'obtenir le contenu et la partie primitive d'un polynôme à coefficients entiers.

Proposition 36 *Si F et G sont deux polynômes de $\mathbb{Z}[x]$,*

$$F \wedge G = (\text{contenu}\, F \wedge \text{contenu}\, G) \times (\text{partie\_primitive}\, F \wedge \text{partie\_primitive}\, G).$$

La fonction suivante retourne un pgcd P de deux polynômes F_0 et G_0 de $\mathbb{Z}[x]$. Ce pgcd est aussi bien un pgcd dans $\mathbb{Z}[x]$ que dans $\mathbb{Q}[x]$. La fonction s'arrête parce que les nombres premiers malchanceux sont en nombre fini.

function $\text{pgcd}(F_0, G_0)$
begin
  p_0 := $\text{contenu}(F_0) \wedge \text{contenu}(G_0)$
  F := $\text{partie\_primitive}(F_0)$
  G := $\text{partie\_primitive}(G_0)$
  p_1 := $\text{coefficient\_dominant}(F) \wedge \text{coefficient\_dominant}(G)$

```
  do
    n := un nouveau nombre premier supérieur à 2 × p_1 × min(2^(deg F) H(F), 2^(deg G) H(G))
         (on remarque que n ne divise pas p_1)
    P_n := (F mod n) ∧ (G mod n)   unitaire, calculé dans (ℤ/nℤ)[x]
    P' := (p_1 P_n) mod n   (prendre des représentants signés)
    P' := partie_primitive(P')   (P' est vu comme un polynôme de ℤ[x])
  while P' ne divise pas F ou P' ne divise pas G dans ℤ[x]
  return p_0 P'
end
```

4.3.8 Une seconde méthode, fondée sur le théorème chinois

L'idée consiste à calculer plusieurs pgcd dans $(\mathbb{Z}/n\mathbb{Z})[x]$, pour différentes valeurs de n et à les combiner entre eux avec le théorème chinois. Dans un langage de programmation traditionnel, on choisirait des nombres premiers d'une taille inférieure à celle d'un mot machine (inférieure à 32 bits sur une machine disposant d'une arithmétique câblée sur 32 bits). On ne combine par le théorème chinois que des pgcd modulaires de même degré. On choisit des nombres premiers qui ne divisent pas le coefficient dominant de F ou celui de G de façon à s'assurer que les degrés des pgcd modulaires soient supérieurs ou égaux à ceux de P. Dans le cas de deux pgcd modulaires de degrés différents, on est donc certain que celui de plus grand degré correspond à un nombre premier malchanceux et on l'oublie. On teste à chaque tour si on a obtenu le pgcd recherché par un test de division exacte dans $\mathbb{Z}[x]$ des polynômes F et G. On n'utilise donc pas la borne de Landau et Mignotte.

La fonction suivante retourne un pgcd P de deux polynômes F_0 et G_0 de $\mathbb{Z}[x]$. Ce pgcd est aussi bien un pgcd dans $\mathbb{Z}[x]$ que dans $\mathbb{Q}[x]$.

Les calculs s'arrêtent parce que les nombres premiers malchanceux sont en nombre fini. Les nombres premiers n choisis sont multipliés entre eux dans la variable m. Pour chacun de ces nombres premiers, un pgcd modulaire P_n est calculé. La variable P_m contient le résultat de la combinaison, par le théorème chinois, des pgcd modulaires P_n. La variable $\overline{P}$ contient un polynôme de $\mathbb{Z}[x]$ correspondant à P_n. L'algorithme détecte très rapidement les polynômes premiers entre eux.

```
function pgcd(F_0, G_0)
begin
  p_0 := contenu(F_0) ∧ contenu(G_0)
  F := partie_primitive(F_0)
  G := partie_primitive(G_0)
  p_1 := coefficient_dominant(F) ∧ coefficient_dominant(G)
  n := un nombre premier qui ne divise pas p_1
  P_n := (F mod n) ∧ (G mod n)   unitaire, calculé dans (ℤ/nℤ)[x]
  if P_n = 1 then
    return p_0
  fi
```

m := n
P_m := P_n
$\overline{P}$:= $\mathrm{partie\_primitive}((p_1\, P_m) \mod m)$
(utiliser des représentants signés pour le calcul modulaire, calculer le contenu dans $\mathbb{Z}[x]$)
while $\overline{P}$ ne divise pas F ou $\overline{P}$ ne divise pas G dans $\mathbb{Z}[x]$ do
n := un nouveau nombre premier qui ne divise pas p_1
P_n := $(F \mod n) \wedge (G \mod n)$ unitaire, calculé dans $(\mathbb{Z}/n\mathbb{Z})[x]$
if $P_n = 1$ then
return p_0
elif $\deg P_n < \deg P_m$ then
m := n
P_m := P_n
$\overline{P}$:= $\mathrm{partie\_primitive}((p_1\, P_n) \mod n)$
elif $\deg P_n = \deg P_m$ then
r := $n \times m$
calculer avec le théorème chinois $P_r \in (\mathbb{Z}/r\mathbb{Z})[x]$ tel que
$P_r = P_n \mod n$
$P_r = P_m \mod m$
m := r
P_m := P_r (noter que P_r est unitaire)
$\overline{P}$:= $\mathrm{partie\_primitive}((p_1\, P_m) \mod m)$
fi
od
return $p_0\, \overline{P}$
end

Chapitre 5

La méthode de Newton

Dans ce chapitre, on présente la méthode numérique de résolution d'équations la plus utilisée : la méthode de Newton.

5.1 Une équation en une variable

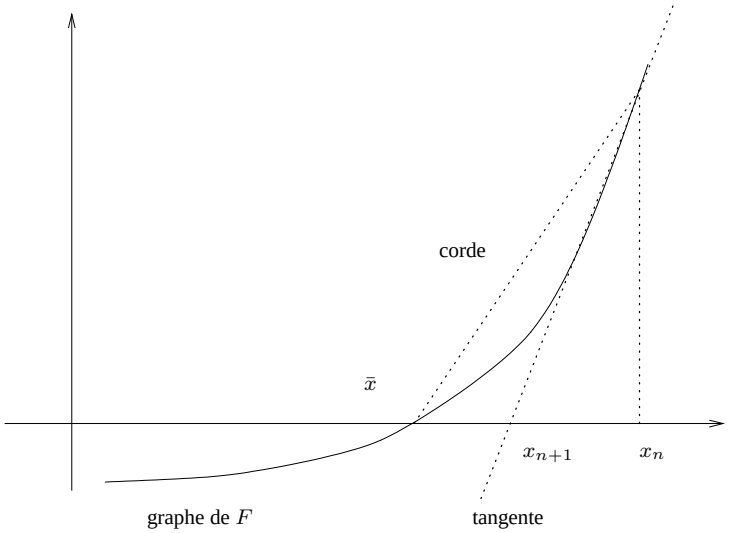
On considère une équation en une variable $F(x) = 0$. On cherche à calculer une approximation $\bar{x}$ d'une racine de l'équation à partir d'une première estimation a.

FIG. 5.1 – Isaac Newton en 1726 (1643–1727)

La méthode de Newton consiste à calculer une suite de points (x_n) à partir de a en espérant que la suite converge vers $\bar{x}$.

$$x_0 = a, \quad x_{n+1} = x_n - \frac{F(x_n)}{F'(x_n)}.$$

On arrête les calculs dès que $|F(x_n)| < \varepsilon$ où ε est un réel positif fixé à l'avance. Graphiquement, x_{n+1} est donné par l'intersection de l'axe des x avec la tangente à la courbe en $(x_n,\ F(x_n))$.



5.1.1 Justification

Soient $\bar{x}$ une racine de $F(x) = 0$ et x_n un réel proche de $\bar{x}$. Notons h la différence $\bar{x} - x_n$ de telle sorte que $\bar{x} = x_n + h$. Alors, en appliquant la formule du développement limité de $F(x)$ en x_n on obtient :

$$0 = F(\bar{x}) = F(x_n + h) = F(x_n) + h\,F'(x_n) + \cdots$$

Comme h est petit, on néglige les termes cachés dans les points de suspension et on trouve :

$$F(x_n) + h\,F'(x_n) \simeq 0$$

et donc $h \simeq -F(x_n)/F'(x_n)$. En reportant cette relation dans la formule $\bar{x} = x_n + h$ on trouve une approximation de $\bar{x}$ qu'on prend comme valeur pour x_{n+1} :

$$x_{n+1} = x_n - \frac{F(x_n)}{F'(x_n)}.$$

5.1.2 Exemple

On applique la méthode de Newton pour calculer la racine positive de l'équation $x^2 - 4 = 0$, c'est–à–dire le nombre 2. On prend pour valeur initiale $x_0 = 3$.

```
Digits := 30:
F := x -> x^2 - 4;
                                        2
                                F := x -> x  - 4
Fprime := D(F);
                               Fprime := x -> 2 x
x[0] := 3.:
for n from 0 to 4 do
    x[n+1] := x[n] - F(x[n])/Fprime(x[n])
```

```
od;
                    x[1] := 2.16666666666666666666666666667

                    x[2] := 2.00641025641025641025641025641

                    x[3] := 2.00001024002621446710903579913

                    x[4] := 2.00000000002621440000017179869

                    x[5] := 2.00000000000000000000017179869
```

La vitesse de convergence est impressionnante : le nombre de décimales correctes double approximativement à chaque itération. On verra ci–dessous que c'est lié au fait que 2 est une racine simple de l'équation utilisée.

Appliquons maintenant la méthode de Newton à l'équation $(x-2)^2 = 0$ dont 2 est racine double. La vitesse de convergence est nettement moins bonne !

```
F := x -> (x-2)^2;
                                          2
                              F := x -> (x - 2)
Fprime := D(F);
                            Fprime := x -> 2 x - 4

x[0] := 3.:
for n from 0 to 4 do
    x[n+1] := x[n] - F(x[n])/Fprime(x[n])
od;
                    x[1] := 2.50000000000000000000000000000

                    x[2] := 2.25000000000000000000000000000

                    x[3] := 2.12500000000000000000000000000

                    x[4] := 2.06250000000000000000000000000

                    x[5] := 2.03125000000000000000000000000
```

5.1.3 Convergence de la méthode

Dans cette section, on utilise MAPLE pour analyser théoriquement la relation entre la vitesse de convergence et la simplicité de la racine. La méthode de Newton peut diverger ou boucler et, même dans le cas où elle converge, elle peut ne pas converger vers la racine $\bar{x}$ désirée. Cependant, lorsqu'on étudie l'ordre d'une méthode numérique de résolution, on considère toujours une racine $\bar{x}$ et on suppose que la méthode converge bien vers elle. Voici une autre remarque importante : dans les méthodes numériques de résolution d'équations, il faut distinguer les erreurs de méthode des erreurs d'arrondis. On ne s'intéresse dans cette section qu'aux erreurs de méthode (erreurs qui persistent même si on mène tous les calculs avec des nombres exacts, ce qui est possible en MAPLE). Les erreurs dues aux arrondis sont beaucoup plus difficiles à estimer.

On définit l'erreur à l'étape n par $e_n \underset{\text{def}}{=} x_n - \bar{x}$.

Définition 30 *Une méthode de résolution d'équations numériques est dite d'ordre p si*

$$\lim_{n \to +\infty} \frac{e_{n+1}}{e_n^p} = \text{cste non nulle.}$$

Dire qu'une méthode est d'ordre un, c'est dire que le nombre de décimales correctes augmente linéairement avec n : à chaque étape, on gagne k nouvelles décimales correctes (pour une certaine constante k). Dire qu'une méthode est d'ordre deux, c'est dire que le nombre de décimales correctes augmente quadratiquement avec n : à chaque étape, le nombre de décimales correctes double (approximativement). Dans ce qui suit, on suppose que la méthode converge bien vers une racine $\bar{x}$.

Proposition 37 *Si $\bar{x}$ est une racine simple de F alors la méthode de Newton est d'ordre deux (au moins) sinon la méthode est d'ordre un.*

On montre avec MAPLE que la méthode de Newton est d'ordre deux dans le cas d'une racine simple. On utilise ici les possibilités de calcul symbolique du logiciel pour mener une « démonstration assistée par ordinateur ».

Comme x_n est proche de $\bar{x}$, on peut approximer le graphe de F par une parabole.

```
F := x -> a*x^2 + b*x + c;
```

$$F := x \to a\,x^2 + b\,x + c$$

```
Fp := D(F);
```

$$Fp := x \to 2\,a\,x + b$$

```
xbar := (- b - sqrt (b^2-4*a*c))/(2*a);
```

$$xbar := \frac{-b - \left(b^2 - 4\,a\,c\right)^{1/2}}{a}\,\frac{1}{2}$$

On exprime x_{n+1} en fonction de x_n.

```
x(n+1) := x(n) - F(x(n))/Fp(x(n));
```

$$x(n+1) := x(n) - \frac{a\,x(n)^2 + b\,x(n) + c}{2\,a\,x(n) + b}$$

On porte la dynamique sur l'erreur : on cherche à exprimer e_{n+1} en fonction de e_n :

```
e(n+1) := x(n+1) - xbar;
```

```
                                2                                2         1/2
                          a x(n)  + b x(n) + c       -b - (b  - 4 a c)
      e(n + 1) := x(n) - -------------------- - 1/2 --------------------
                            2 a x(n) + b                     a
e(n+1) := subs (x(n) = e(n) + xbar, e(n+1)):
e(n+1) := simplify (e(n+1));
                                        2
                                     e(n)  a
                  e(n + 1) := - ---------------------------
                                        2         1/2
                                -2 e(n) a + (b  - 4 a c)
```

Si la racine est simple, c'est–à–dire si $b^2 - 4\,a\,c > 0$ alors le rapport e_{n+1}/e_n^2 tend vers une constante non nulle quand e_n tend vers zéro.

```
simplify (e(n+1)/e(n)^2);
                                    a
                          - --------------------------
                                     2         1/2
                            -2 e(n) a + (b  - 4 a c)
subs (e(n) = 0, %);
                                    a
                               - --------------
                                   2         1/2
                                 (b  - 4 a c)
```

5.2 Deux équations en deux variables

Le schéma de Newton se généralise aux systèmes de deux équations en deux variables et, plus généralement, aux systèmes de n équations en n variables.

On considère un système de deux équations de deux variables réelles :

$$F(x,\, y) = 0, \quad G(x,\, y) = 0.$$

La suite de Newton consiste à calculer une suite de points $(x_n,\, y_n)$ à partir d'un point donné $(x_0,\, y_0)$. La relation de récurrence pour le schéma de Newton en deux variables s'écrit :

$$x_{n+1} = x_n + h, \quad y_{n+1} = y_n + \ell$$

où h et ℓ sont deux réels qui sont recalculés à chaque itération. Ces réels sont en fait la solution du système d'équations suivant, qui dépend de x_n et de y_n et qu'il suffit donc de résoudre à chaque itération. La matrice des coefficients est souvent appelée la matrice *jacobienne* du système à résoudre :

$$\begin{pmatrix} \dfrac{\partial F}{\partial x}(x_n,\, y_n) & \dfrac{\partial F}{\partial y}(x_n,\, y_n) \\ \dfrac{\partial G}{\partial x}(x_n,\, y_n) & \dfrac{\partial G}{\partial y}(x_n,\, y_n) \end{pmatrix} \cdot \begin{pmatrix} h \\ \ell \end{pmatrix} = - \begin{pmatrix} F(x_n,\, y_n) \\ G(x_n,\, y_n) \end{pmatrix}.$$

5.2.1 Justification

Soient $(\bar{x},\, \bar{y})$ une solution de ce système et $(x_n,\, y_n)$ un couple de réels proche de $(\bar{x},\, \bar{y})$. Notons h et ℓ les différences $\bar{x} - x_n$ et $\bar{y} - y_n$ de telle sorte que $\bar{x} = x_n + h$ et $\bar{y} = y_n + \ell$. Alors, en appliquant la formule du développement limité de $F(x,\, y)$ et de $G(x,\, y)$ en $(x_n,\, y_n)$ on obtient une formule :

$$0 = F(\bar{x},\, \bar{y}) = F(x_n + h,\, y_n + \ell) = F(x_n,\, y_n) + h\,\frac{\partial F}{\partial x}(x_n,\, y_n) + \ell\,\frac{\partial F}{\partial y}(x_n,\, y_n) + \cdots$$

$$0 = G(\bar{x},\, \bar{y}) = G(x_n + h,\, y_n + \ell) = G(x_n,\, y_n) + h\,\frac{\partial G}{\partial x}(x_n,\, y_n) + \ell\,\frac{\partial G}{\partial y}(x_n,\, y_n) + \cdots$$

Comme h et ℓ sont petits, on néglige les termes cachés dans les points de suspension et on trouve :

$$0 \simeq F(x_n,\, y_n) + h\,\frac{\partial F}{\partial x}(x_n,\, y_n) + \ell\,\frac{\partial F}{\partial y}(x_n,\, y_n),$$

$$0 \simeq G(x_n,\, y_n) + h\,\frac{\partial G}{\partial x}(x_n,\, y_n) + \ell\,\frac{\partial G}{\partial y}(x_n,\, y_n)\cdot$$

Transformons les $\simeq$ en $=$ et écrivons ce système matriciellement. On obtient un système de deux équations linéaires dont les inconnues sont les réels h et ℓ. Il s'agit du système donné en début de section.

5.2.2 Exemple

Les commandes MAPLE suivantes montrent comment on peut appliquer la méthode de Newton pour déterminer une solution du système :

$$x^2 + y^2 - 10 = 0, \quad x - 3\,y = 0.$$

On peut montrer que ce système a deux solutions réelles, qui sont $(x, y) = (3, 1)$ et $(-3, -1)$. L'itération de Newton converge vers la première solution. La vitesse de convergence est quadratique.

Le calcul de la matrice jacobienne est effectué par la fonction *Jacobian* du paquetage *VectorCalculus*. Pour les calculs d'algèbre linéaire, on utilise le paquetage *LinearAlgebra* de MAPLE. Pour la résolution du système linéaire, on construit une matrice M « générique » en bordant la matrice jacobienne avec le vecteur colonne de coordonnées $-F$ et $-G$. On la spécialise à chaque itération avec les valeurs de x_n et de y_n.

```
with (LinearAlgebra):
with (VectorCalculus,Jacobian):
F := x^2 + y^2 - 10:
G := x - 3*y:

M := < Jacobian ([F, G], [x, y]) | - <F, G> >;
```

```
                          [                2    2     ]
                    M := [2 x    2 y    -x  - y  + 10]
                          [                          ]
                          [ 1     -3       -x + 3 y   ]
```

```
x[0] := 2.:
y[0] := 2.:
for n from 0 to 1 do
    Mn := subs (x=x[n], y=y[n], M):
    hl := LinearSolve (Mn):
    x[n+1] := x[n] + hl[1]:
    y[n+1] := y[n] + hl[2]:
od;
```

```
                                [4.    4.    2.]
                           Mn := [              ]
                                 [1     -3    4.]

                                    [ 1.37500000000000000 ]
                               hl := [                     ]
                                    [-0.87500000000000000]

                                 x[1] := 3.375000000

                                 y[1] := 1.125000000

                        [6.750000000    2.250000000    -2.656250000]
                   Mn := [                                         ]
                        [     1             -3              0.     ]

                                    [-0.354166666666666685]
                               hl := [                     ]
                                    [-0.118055555555555538]

                                 x[2] := 3.020833333

                                 y[2] := 1.006944444
```

Chapitre 6

Introduction au logiciel MAPLE

6.1 Les nombres

Le logiciel MAPLE permet de manipuler différents types de nombres. Pour implanter un type de nombre, la difficulté fondamentale consiste à reconnaître zéro. En effet, un programmeur qui souhaiterait implanter un nouveau type de nombre pourrait toujours réaliser les quatre opérations trivialement (en ne simplifiant jamais les expressions). Par exemple, rien ne force à « simplifier » le produit $\sqrt{2} \cdot \sqrt{3}$ en $\sqrt{6}$. Mais ne pas simplifier les expressions ne fait que reporter les difficultés sur le test d'égalité à zéro puisqu'il faudra bien que ce test reconnaisse que $\sqrt{2} \cdot \sqrt{3} - \sqrt{6}$ est égal à zéro.

6.1.1 Les nombres inexacts

On les appelle aussi « nombres à virgule flottante ». C'est le type de nombre le plus répandu en informatique. Les quatre opérations sont assez simples à implanter. Elles souffrent du problème des erreurs d'arrondi. Le test d'égalité à zéro n'a pas de sens en pratique en raison de ces erreurs. Il est donc souvent remplacé par un test « inférieur à ε ». Une particularité de MAPLE : le nombre de décimales peut être fixé arbitrairement grand.

```
a := sqrt (2.);
                                a := 1.414213562
a^2;
                                   1.999999999
Digits := 30:
sqrt (2.);
                      1.41421356237309504880168872421
```

6.1.2 Les entiers et les rationnels

On rencontre le type « nombre entier » dans tous les langages de programmation mais il s'agit presque tout le temps de nombres entiers de taille limitée. Le logiciel MAPLE fournit des entiers

et des rationnels arbitrairement grands. Ces nombres sont presque toujours représentés de façon interne dans une base de numération. Le test d'égalité à zéro est facile dans une base de numération.

6.1.3 Les nombres irrationnels

Les nombres irrationnels se divisent en nombres algébriques et nombres transcendants.

Définition 31 *Un nombre est dit* algébrique *s'il est racine d'un polynôme en une indéterminée et à coefficients rationnels. Un nombre qui n'est pas algébrique est dit* transcendant.

Les nombres $\sqrt{2}$ et i sont algébriques, tous les rationnels aussi. Les nombres π et e sont transcendants.

Une implantation classique consiste à représenter chaque nombre algébrique α par un polynôme P de degré minimal dont α est racine plus un intervalle (ou un couple d'intervalles dans le cas de nombres algébriques complexes) permettant de distinguer α des autres racines de P. Le test d'égalité à zéro existe mais il est si coûteux que le logiciel ne le met pas systématiquement en œuvre bien qu'il soit implanté.

```
sqrt (2);
                                      1/2
                                     2
sqrt (2)^2;
                                       2

if sqrt (2) * sqrt (3) = sqrt (6) then
        OUI
  else
        NON
  fi;
                                     NON

simplify (sqrt (2) * sqrt (3) - sqrt (6));
                                       0
```

Le nombre imaginaire i est noté *I*. Le nombre π est désigné par (attention à la majuscule) *Pi*.

```
(2+3*I)*(2-3*I);
                                     13

evalf (Pi, 20);
                          3.1415926535897932385
```

6.2 Variables et symboles

En MAPLE, un symbole est une variable (au sens informatique du terme) à laquelle aucune valeur n'est affectée. On transforme un symbole en une variable en lui affectant une valeur. On

transforme une variable en un symbole en le « désaffectant », ce qui peut se réaliser en lui affectant son propre identificateur entre apostrophes (ou « quotes »). La présence d'apostrophes autour d'un identificateur empêche son évaluation. La variable spéciale « % » contient le résultat de la dernière commande exécutée.

```
(a+1)^2;
                                        2
                                   (a + 1)

a := 4;
                                    a := 4

(a+1)^2;
                                      25
a := 'a';
                                    a := a

(a+1)^2;
                                        2
                                   (a + 1)

%;
                                        2
                                   (a + 1)
```

6.3 Procédures et fonctions

On ne fait pas de différence en MAPLE entre les procédures et les fonctions. Les procédures peuvent donc retourner un résultat. Une procédure est un objet comme un autre, qui peut être affecté à une variable. C'est d'ailleurs ainsi qu'on les nomme.

Par convention, la valeur retournée par une procédure est la valeur de la dernière instruction exécutée lors de l'appel. Pour forcer l'exécution d'un appel de procédure à s'arrêter et à retourner une valeur, on peut utiliser l'instruction *return*.

On mentionne aussi la fonction *error*, paramétrée par une chaîne de caractères (délimitée par des guillemets) et qui permet d'interrompre un calcul en affichant le message.

Une procédure peut utiliser des variables *locales* et des variables *globales*. Les variables globales sont les variables qui sont immédiatement accessibles lorsqu'on utilise MAPLE interactivement. Les variables locales d'une procédure sont créées à chaque appel à la procédure (elles sont locales à un appel de procédure plutôt qu'à une procédure). En règle générale, elles disparaissent à la fin de l'appel (voir toutefois la section 6.3.3). Les déclarations de variables doivent être placées juste après l'entête de la procédure. Lorsqu'une variable n'est pas déclarée, MAPLE infère son type de la façon suivante : si la première opération effectuée sur la variable modifie son contenu, la variable est considérée comme locale sinon elle est considérée comme globale. On suggère de déclarer explicitement les variables.

On rappelle qu'en programmation, on distingue les *paramètres formels* d'une procédure (dont les identificateurs sont énumérés dans l'entête de la procédure, derrière le mot–clef *proc*) des pa-

ramètres effectifs des *appels de procédures*. Au moment de l'appel à la procédure, chaque paramètre formel prend pour valeur l'un des paramètres effectifs.

6.3.1 Exemple

On définit une procédure paramétrée par un réel a, un entier n et qui retourne a^n. On l'affecte à la variable *puissance*. La procédure emploie deux variables locales : k et *resultat*. Les paramètres formels sont a et n. Le cas $a = 0$ est géré à part.

```
puissance := proc (a, n)
    local k, resultat;
    if a = 0 then
        return (0)
    fi;
    resultat := 1;
    for k from 1 to n do
        resultat := resultat * a
    od;
    resultat
end:
```

Voici deux appels à la procédure puissance. Deux paramètres effectifs sont fournis à chaque appel.

```
puissance (0,0);
                                    0

puissance (3,4);
                                   81
```

6.3.2 Retourner une valeur en l'affectant à un paramètre

En règle générale, les paramètres sont passés par valeur et il est impossible d'affecter une valeur à un paramètre formel dans le corps d'une procédure.

Cette règle admet une exception : si la valeur du paramètre effectif est un identificateur de variable alors il est possible d'affecter une valeur au paramètre formel correspondant : c'est la variable dont le nom a été passé comme paramètre effectif qui reçoit la valeur. Remarque : après qu'on a affecté une valeur à un tel paramètre dans un corps de procédure, il n'est pas possible de consulter le contenu de ce paramètre (les paramètres formels sont gérés différemment des variables locales en MAPLE).

Ce mécanisme est mis en œuvre par certaines fonctions prédéfinies telles que *rem*, *quo* ou *coeffs*. En voici un exemple.

```
reste := proc (a, b, quotient)
    local q, r;
    q := iquo (a, b);
    r := irem (a, b);
    quotient := q;      # affecte q à la var. dont le nom est dans quotient
    r                   # retourne r
end:
```

Dans l'appel de procédure ci–dessous, on a placé l'identificateur *quot* entre apostrophes pour éviter qu'il ne soit remplacé par sa valeur au moment de l'appel. On suggère d'utiliser systématiquement des apostrophes non seulement pour la raison énoncée ci–dessus mais aussi parce qu'elles attirent l'attention du lecteur sur l'usage qui est fait du paramètre.

```
reste (7, 3, 'quot');
                                    1
quot;
                                    2
```

6.3.3 Retourner une expression symbolique

Une procédure qui retourne une expression symbolique doit en général s'assurer que chaque symbole figurant dans l'expression est global. Dans le cas contraire, l'utilisateur rencontrera de grandes difficultés pour manipuler l'expression retournée. La présence de « variables locales exportées » (ou plutôt de « symboles locaux exportés ») dans les expressions est une source de bugs considérable.

La fonction suivante retourne une expression en le symbole global x. L'expression retournée se manipule sans difficulté.

```
polynome := proc (n)
    local i;
    global x;
    add ((i+1)*x^i, i=0..n)
end:

p := polynome (3);
                                        2      3
                          p := 1 + 2 x + 3 x  + 4 x

coeff (p, x, 2);
                                    3
```

L'expression retournée par la fonction suivante comporte un symbole local exporté : x. L'expression retournée est beaucoup plus difficile à manipuler ! En effet, le symbole x figurant dans l'expression a été créé lors de l'appel à la fonction *polynome*. Il est différent du symbole global x passé en paramètre à *coeff*.

```
polynome := proc (n)
    local i, x;
    add ((i+1)*x^i, i=0..n)
end:

p := polynome (3);
                                        2      3
                          p := 1 + 2 x + 3 x  + 4 x

coeff (p, x, 2);
                                    0
```

6.3.4 Fonctions définies avec une flèche

Les fonctions dont le corps se réduit à une seule instruction peuvent s'écrire simplement grâce à l'opérateur « flèche ». Les deux implantations de f ci–dessous sont équivalentes.

```
f := proc (x)
    3*x^2 + 2*x + 1
end:

f := x -> 3*x^2 + 2*x + 1:
```

Voici un autre exemple avec une fonction de deux variables.

```
f := proc (x, y)
    3*x^2 + x*y
end:

f := (x,y) -> 3*x^2 + x*y:
```

6.3.5 Fonctions et expressions

Ne pas confondre *fonctions* et *expressions*. Dans l'exemple suivant, f est une fonction et p est une expression. La fonction *D* permet de dériver une fonction, *diff* permet de dériver une expression. On évalue les expressions en utilisant la fonction *subs*. On évalue les fonctions en procédant à des appels de fonctions.

```
f := x -> 2*x^2 + 3*x + 1;
                                      2
                          f := x -> 2 x  + 3 x + 1

p := 2*x^2 + 3*x + 1;
                                    2
                             p := 2 x  + 3 x + 1

f(1);
                                     6

subs (x=1, p);
                                     6

D (f);
                                x -> 4 x + 3

diff (p, x);
                                   4 x + 3
```

Pour convertir une fonction en une expression, il suffit d'évaluer la fonction en fournissant un symbole comme paramètre effectif. La fonction prédéfinie *unapply* permet de convertir une expression en une fonction.

```
f (y);
                                    2
                                 2 y  + 3 y + 1
unapply (p, x);
                                     2
                               x -> 2 x  + 3 x + 1
```

6.4 Structures de données

6.4.1 Séquences, listes et ensembles

Définition 32 *Une* séquence *est une suite d'objets séparés par des virgules. La virgule est l'opérateur de concaténation des séquences. La séquence vide est nommée NULL.*

La fonction *seq* permet de construire des séquences.

```
s1 := 3, x + 1, 2;
                                 s1 := 3, x + 1, 2

s2 := 2, x*y;
                                   s2 := 2, x y

seq (i^2, i = 1 .. 5);
                                  1, 4, 9, 16, 25

s2, NULL;
                                     2, x y

s3 := s1, s2;
                           s3 := 3, x + 1, 2, 2, x y
```

Les séquences ont un défaut : il n'est pas possible de construire des séquences de séquences . . . sauf à délimiter les sous–séquences avec des marqueurs de début et de fin. C'est à cette fin qu'on été définis les listes et les ensembles.

Définition 33 *Une* liste *est une séquence délimitée par des crochets.*

La fonction *op* permet d'obtenir les opérandes d'une liste (la séquence de ses éléments). Le fonction *nops* permet d'obtenir le nombre d'opérandes (d'éléments) d'une liste. La fonction *map* permet d'appliquer une fonction unaire à tous les éléments d'une liste. Remarque : il est possible d'appliquer des fonctions n–aires aux éléments d'une liste avec *map* ; voir aussi à ce sujet la fonction *zip*.

```
L3 := [ s3 ];
                          L3 := [3, x + 1, 2, 2, x y]

nops (L3);
                                       5
```

```
L3 [2];
                                    x + 1

op (L3);
                              3, x + 1, 2, 2, x y

map (proc (x) x^2 end, L3);
                                      2          2  2
                          [9, (x + 1) , 4, 4, x  y ]
```

Définition 34 *Un* ensemble *est une séquence encadrée par des accolades. Deux ensembles qui contiennent les mêmes éléments sont égaux.*

Les ensembles MAPLE ressemblent beaucoup aux ensembles mathématiques donnés en extension : l'ordre des éléments n'est pas spécifié et chaque élément n'apparaît qu'une fois. Lorsqu'on convertit une séquence en ensemble, l'ordre des éléments de la séquence peut donc être modifié et les doublons sont supprimés. Une précision sur l'ordre des éléments d'un ensemble : si les variables E et F contiennent deux ensembles égaux alors l'ordre des éléments de E et de F est le même mais cet ordre peut changer d'une session MAPLE à une autre. Les fonctions *op*, *nops* et *map* s'appliquent aussi aux ensembles.

```
E3 := { s3 };
                           E3 := {2, 3, x + 1, x y}

nops (E3);
                                     4

op (E3);
                              2, 3, x + 1, x y

E3 [2];
                                     3
```

6.4.2 Équations et intervalles

L'opérateur .. permet de former des intervalles. On accède aux membres gauche et droit d'une équation ou d'un intervalle grâce aux fonctions prédéfinies *lhs* (pour *left handside*) et *rhs* (pour *right handside*). Des intervalles sont fournis en paramètre aux fonctions *seq*, *add*, *mul*, ... Des équations sont fournies en paramètre aux fonctions de la famille de *solve*.

```
add (i, i = 1..5);
                                    15

mul (i, i = 1..5);
                                   120

eq := 2*x+3*y = 7;
```

```
eq := 2 x + 3 y = 7
```

```
solve (eq);
```

$$\{x = -\frac{3\,y}{2} + 7/2,\ y = y\}$$

```
lhs (eq);
```

$$2\,x + 3\,y$$

```
rhs (eq);
```

$$7$$

6.4.3 Polynômes

Les fonctions prédéfinies `expand` (développer) et `collect` (regrouper les coefficients) permettent d'écrire un polynôme de plusieurs façons.

```
P := (x - a)^2 + 3*x + a;
```

$$P := (x - a)^2 + 3\,x + a$$

```
expand (P);
```

$$x^2 - 2\,x\,a + a^2 + 3\,x + a$$

```
collect (P, x);
```

$$x^2 + (-2\,a + 3)\,x + a^2 + a$$

```
collect (P, a);
```

$$a^2 + (-2\,x + 1)\,a + x^2 + 3\,x$$

Ces fonctions ont une utilité supplémentaire : elles permettent de s'assurer du degré, du coefficient dominant etc ... d'un polynôme. L'exemple ci–dessous montre que la fonction *degree* peut retourner un faux résultat, lorsqu'on l'applique à un polynôme qui n'a pas été préalablement réécrit avec l'une de ces fonctions.

```
Q := (x-1)^2 - x^2 + 2*x - 1;
```

$$Q := (x - 1)^2 - x^2 + 2\,x - 1$$

```
degree (Q,x);
```

$$2$$

```
degree (expand (Q), x);
```

$$-\text{infinity}$$

La fonction *indets* détermine l'ensemble des symboles dont dépend un polynôme. La fonction *lcoeff* permet d'obtenir le coefficient dominant d'un polynôme. La fonction *coeff* permet d'obtenir le coefficient d'un symbole à un certain degré dans un polynôme. On remarque que les monômes n'apparaissent pas nécessairement pas degré décroissant. La fonction à effet de bord *sort* ordonne les monômes par degré décroissant.

```
P := expand ((x-y)^2 +3*x - z);
                                2            2
                       P := x  - 2 x y + y  + 3 x - z

indets (P);
                                  {x, z, y}

v := indets (P) [1];
                                   v := x

lcoeff (P, v);
                                      1

coeff (P, v, 0);
                                    2
                                   y  - z

sort (P);
                             2            2
                            x  - 2 x y + y  + 3 x - z
```

La fonction *coeffs* permet d'obtenir la séquence des coefficients d'un polynôme. Les coefficients ne sont pas triés du tout. Pour obtenir la correspondance avec les termes (les monômes), on utilise un troisième paramètre qui reçoit la séquence des termes rangés dans le même ordre que les coefficients. La fonction *zip* permet d'appliquer une fonction binaire à tous les éléments de deux listes.

```
P := 3*x^3 - 7*x^2 - 11*x + 237;
                                  3      2
                         P := 3 x  - 7 x  - 11 x + 237

koeffs := [ coeffs (P, x, 'termes') ];
                          koeffs := [237, -11, -7, 3]

termes := [ termes ];
                                           2   3
                           termes := [1, x, x , x ]

zip ( proc (x,y) [x,y] end, koeffs, termes );
                                             2        3
                   [[237, 1], [-11, x], [-7, x ], [3, x ]]
```

6.5 Structures de contrôle

6.5.1 Les boucles for

Voici plusieurs boucles *for* qui affectent toutes à une variable S la somme de tous les éléments d'une liste L.

```
S := 0;
for n from 1 to nops (L) do
    S := S + L[n]
od;

S := 0;
for n from nops (L) to 1 by -1 do
    S := S + L[n]
od;

S := 0;
for x in L do
    S := S + x
od;
```

Définition 35 *(invariant de boucle* for*)*
Un invariant de boucle for *est une propriété vraie au début de chaque itération.*

La propriété « m est le plus grand des éléments de L d'indice strictement inférieur à k » est un invariant de la boucle de l'exemple de la fonction *maximum* ci–dessous. La boucle s'arrête à la fin de l'itération $k = \mathit{nops}\,(L)$ et donc en quelque sorte au début de l'itération « fictive » $k = \mathit{nops}\,(L) + 1$. D'après l'invariant, m contient le plus grand élément de L.

6.5.2 La structure if

Voici une implantation d'une fonction définie par morceaux.

```
f := proc (x)
    local resultat;
    if x <= 0 then
        resultat := 0
    elif x <= 2 then
        resultat := x
    else
        resultat := x^2 - 2
    fi;
    resultat
end;
```

La fonction suivante retourne le plus grand élément d'une liste de nombres.

```
maximum := proc (L)
    local m, k;
    m := L [1];
    for k from 2 to nops (L) do
        if m < L [k] then
            m := L [k]
        fi
    od;
    m
end;
```

6.5.3 La boucle while

La boucle *while* suivante affecte à une variable S la somme des éléments d'une liste.

```
S := 0;
n := 1;
while n <= nops (L) do
    S := S + L[n];
    n := n+1
od;
```

La fonction suivante est paramétrée par un objet x, une liste L et retourne *true* si x appartient à L et *false* sinon.

```
appartient := proc (x, L)
    local k, trouve;
    trouve := false;
    k := 1;
    while k <= nops (L) and not trouve do
        if x = L[k] then
            trouve := true
        fi;
        k := k + 1
    od;
    trouve
end
```

Définition 36 *(invariant de boucle while)*
Un invariant de boucle while *est une propriété vraie à chaque fois que la condition du* while *est évaluée.*

En particulier, un invariant de boucle *while* est vrai lorsque la condition s'évalue en *faux*, c'est–à–dire quand la boucle s'arrête. La propriété « *trouve* vaut *vrai* si et seulement si x est égal à l'un des éléments de L d'indice strictement inférieur à k » est un invariant de la boucle de la fonction *appartient* ci–dessus.

6.6 Éléments d'algèbre de Boole

En termes mathématiques, les conditions apparaissant dans les « if » et les « while » sont des expressions *booléennes*, du nom du mathématicien George Boole.

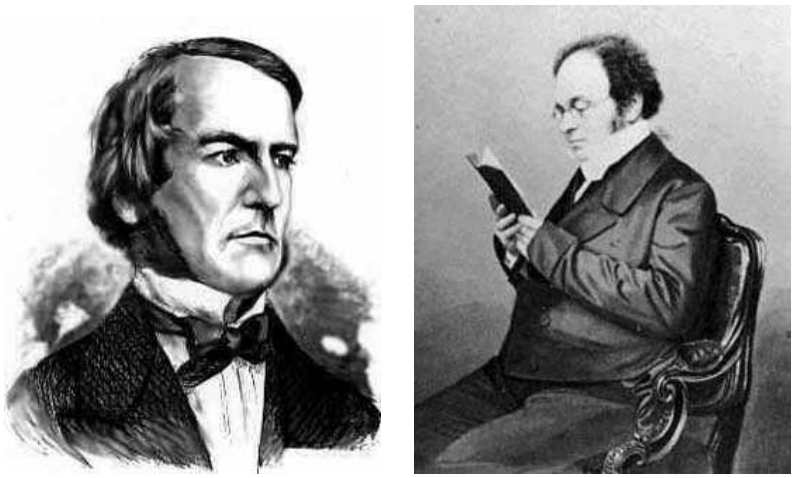


FIG. 6.1 – George Boole (1815–1864) et Augustus De Morgan (1806–1871).

Définition 37 *Une* expression booléenne *est une expression qui s'évalue en* vrai *ou* faux

Les valeurs *vrai* et *faux* se notent *true* et *false* en MAPLE. Par exemple,

$$x \geq 3, \quad (x \geq 3 \ \ et \ \ 3\,x \neq 2), \quad x \text{ est un nombre pair}$$

sont des expressions booléennes. Par contre

$$x + 7, \quad \text{la moitié d'un nombre pair}$$

ne sont pas des expressions booléennes. Les opérateurs $\leq$, $<$, $=$, $\neq$, $\geq$, $>$ sont appelés *opérateurs relationnels*. Ils servent à construire des expressions booléennes élémentaires. Ils sont notés en MAPLE $<=$, $<$, $=$, $<>$, $>=$ et $>$. Les opérateurs *et*, *ou*, *non* sont appelés *opérateurs logiques*. Ils servent à construire des expressions booléennes compliquées à partir d'expressions booléennes élémentaires. Ils sont notés *and*, *or* et *not* en MAPLE. Voici leur table de vérité :

and	vrai	faux
vrai	vrai	faux
faux	faux	faux

or	vrai	faux
vrai	vrai	vrai
faux	vrai	faux

not	
vrai	faux
faux	vrai

Les trois opérateurs logiques décrits ci–dessus sont liés par les *lois de De Morgan*, du nom du mathématicien Augustus De Morgan.

Proposition 38 *(lois de De Morgan)*
Quelles que soient les expressions booléennes a *et* b *on a*

$$\begin{aligned} \textit{non } (a \textit{ et } b) &= (\textit{non } a) \textit{ ou } (\textit{non } b) \\ \textit{non } (a \textit{ ou } b) &= (\textit{non } a) \textit{ et } (\textit{non } b) \end{aligned}$$

Exemple. Reprenons l'exemple de la fonction *appartient*. On cherche, en effectuant une boucle, si un élément appartient à une liste. La boucle s'arrête si

$$\underline{k \text{ dépasse le nombre d'éléments de } L} \text{ ou } \underline{\text{le booléen } \textit{trouve} \text{ vaut } \textit{vrai}}.$$

La condition du *while* est la négation de l'expression booléenne ci–dessus. Elle s'obtient grâce aux lois de De Morgan : les calculs continuent tant que

$$\underline{k \text{ ne dépasse pas le nombre d'éléments de } L} \text{ et } \underline{\text{le booléen } \textit{trouve} \text{ vaut } \textit{faux}}.$$

Remarque. L'opérateur logique *and* de MAPLE n'évalue pas son second opérande si le premier s'évalue en *faux*. Cette particularité permet d'écrire la fonction *appartient* de la façon suivante.

```
appartient := proc (x, L)
    local k;
    k := 1;
    while k <= nops (L) and x <> L[k] do
        k := k + 1
    od;
    evalb (k <= nops (L))
end
```

Cette écriture aurait été incorrecte dans la cas d'un *and* évaluant systématiquement tous ses opérandes puisque la fonction aurait cherché à déterminer $L[k]$ pour $k > \textit{nops}\,(L)$.

6.7 Le paquetage LinearAlgebra

Le paquetage *LinearAlgebra* permet de faire de l'algèbre linéaire en MAPLE. Le contenu de ce paquetage n'est pas immédiatement disponible au lancement de MAPLE. La commande *with* permet de le charger en mémoire.

```
with (LinearAlgebra):

Matrix (2,3);
                                [0    0    0]
                                [           ]
                                [0    0    0]

Vector[column] (3);
                                     [0]
                                     [ ]
                                     [0]
                                     [ ]
                                     [0]

Vector[row] (4);
                                [0, 0, 0, 0]
```

Le paquetage *LinearAlgebra* offre une notation alternative permettant de définir des matrices et des vecteurs. L'opérateur « virgule » permet d'ajouter des lignes à une matrice. L'opérateur « barre verticale » permet d'ajouter des colonnes (penser à la notation traditionnelle utilisée pour border une matrice avec un vecteur).

```
# Les éléments sont ajoutés les uns après les autres en formant à
# chaque fois une nouvelle ligne : on obtient un vecteur colonne.
<1,2,3>;
```

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
# Les éléments sont ajoutés les uns après les autres en formant à
# chaque fois une nouvelle colonne : on obtient un vecteur ligne.
<1|2|3>;
```

$$[1, 2, 3]$$

```
# L'opérateur , permet de border une matrice avec un vecteur ligne
M := <<1|2|3>, <4|5|6>>;
```

$$M := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
W := <7|8|9>;
```

$$W := [7, 8, 9]$$

```
<M, W>;
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
# L'opérateur | permet de border une matrice avec un vecteur colonne
W := <10, 11>;
```

$$W := \begin{bmatrix} 10 \\ 11 \end{bmatrix}$$

```
<M | W>;
```

$$\begin{bmatrix} 1 & 2 & 3 & 10 \\ 4 & 5 & 6 & 11 \end{bmatrix}$$

L'opérateur « + » permet d'additionner deux matrices. L'opérateur « * » permet de multiplier une matrice par un scalaire. L'opérateur « . » permet de multiplier deux matrices entre elles. L'opérateur

« accent circonflexe » permet d'élever une matrice à une certaine puissance. Lorsque l'exposant vaut -1 on obtient l'inverse de la matrice. Il est souvent utile en programmation de pouvoir déterminer les dimensions d'une matrice ou d'accéder à des éléments à partir de leurs indices de ligne et de colonne.

```
Dimension (M);
                                    2, 3

RowDimension (M);
                                      2

ColumnDimension (M);
                                      3
# Élément ligne 1 et colonne 3.
M [1,3];
                                      3
```

FIG. 6.2 – Carl Friedrich Gauss (1777–1855) en 1803.

L'algorithme du pivot de Gauss est accessible de plusieurs façons dans le paquetage. L'algorithme de base est implanté sous le nom *GaussianElimination*. Sa variante, dite de Gauss–Jordan, est implantée sous le nom *ReducedRowEchelonForm*. Une des principales applications du pivot de Gauss est la résolution de systèmes d'équations linéaires. Les fonctions décrites ci–dessus sont donc implicitement appelées par la fonction *LinearSolve*, qu'on présente sur l'exemple ci–dessous :

```
# Le système d'équations à résoudre
S := [  2*x1 -   x2 + 4*x3  = -2,
       -2*x1 + 2*x2 - 3*x3  = 3,
        4*x1 -   x2 + 8*x3  = 1 ]:

# La liste des inconnues
X := [x1, x2, x3]:
```

```
# GenerateMatrix retourne une séquence formée de la
# matrice des coefficients de S suivie du vecteur des seconds membres.
# L'ordre des inconnues dans la liste X donne l'ordre des colonnes
# dans la matrice des coefficients.

# Observer la technique utilisée pour affecter la séquence résultat de
# l'appel de fonction aux deux variable A et b.

A, b := GenerateMatrix (S, X);
                                [ 2    -1     4]  [-2]
                                [              ]  [  ]
                          A, b := [-2     2    -3], [ 3]
                                [              ]  [  ]
                                [ 4    -1     8]  [ 1]

# Ab s'obtient en bordant A avec b.

Ab := <A | b>;
                                  [ 2    -1     4    -2]
                                  [                   ]
                            Ab := [-2     2    -3     3]
                                  [                   ]
                                  [ 4    -1     8     1]

# LinearSolve retourne la solution du système.

V := LinearSolve (Ab);
                                       [19/2]
                                       [    ]
                                  V := [ 5  ]
                                       [    ]
                                       [ -4 ]

# À chaque variable X[i] correspond la valeur V[i]

subs (seq (X[i]=V[i], i=1..3), S);
                             [-2 = -2, 3 = 3, 1 = 1]
```

Index

Bibliographie

[1] Alkiviadis G. Akritas. There is no Uspensky's method. In *proceedings of ISSAC'86*, 1986.

[2] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer Verlag, 2003.

[3] Etienne Bézout. *Cours de mathématiques à l'usage des Gardes du Pavillon et de la Marine*, volume III. Musier, Paris, 1766.

[4] Bruno Buchberger. *An Algorithm for Finding a Basis for the Residue Class Ring of a Zero–Dimensional Polynomial Ideal (German)*. PhD thesis, Math. Inst. Univ. of Innsbruck, Austria, 1965.

[5] George E. Collins and Alkiviadis G. Akritas. Polynomial real root isolation using Descartes'rule of signs. In *proceedings of ISSAC'76*, pages 272–275, Yorktown Heights NY, 1976.

[6] David Cox, John Little, and Donal O'Shea. *Ideals, Varieties and Algorithms. An introduction to computational algebraic geometry and commutative algebra*. Undergraduate Texts in Mathematics. Springer Verlag, New York, 1992.

[7] René Descartes. *Géométrie*. 1636.

[8] Leonard Eugene Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *Am. J. Math.*, 35 :413–422, 1913.

[9] Jean-Charles Faugère. A new efficient algorithm to compute Gröbner bases (F_4). *Journal of Pure and Applied Algebra*, 139 :61–88, 1999.

[10] Jean-Charles Faugère, Patricia Gianni, Daniel Lazard, and Teo Mora. Efficient computation of Gröbner bases by change of orderings. *Journal of Symbolic Computation*, 16 :329–344, 1993.

[11] Eldon Hansen. *Global Optimization Using Interval Analysis*. Marcel Dekker Inc., New York, 1992.

[12] Edmund George Hermann Landau. Sur quelques théorèmes de M. Petrovic relatifs aux zéros des fonctions analytiques. *Bulletin de la Société Mathématique de France*, 33 :251–261, 1905.

[13] Daniel Lazard. Solving Zero–dimensional Algebraic Systems. *Journal of Symbolic Computation*, 13 :117–131, 1992.

[14] Maurice Mignotte. An inequality about factors of polynomials. *Mathematics of Computation*, 28 :1153–1157, 1974.

[15] Maurice Mignotte. *Mathématiques pour le calcul formel*. Presses Universitaires de France, Paris, 1989.

[16] Marc Moreno Maza. *Calculs de Pgcd au–dessus des Tours d'Extensions Simples et Résolution des Systèmes d'Équations Algébriques*. PhD thesis, Université Paris VI, France, 1997.

[17] John J. O'Connor and Edmund F. Robertson. The MacTutor History of Mathematics archive. `http://www-history.mcs.st-andrews.ac.uk/history`, 2006.

[18] Fabrice Rouillier. Solving zerodimensional polynomial systems through the Rational Univariate Representation. Technical report, INRIA, 1998. (number 3426).

[19] Fabrice Rouillier and Paul Zimmermann. Efficient isolation of a polynomial real roots. Technical report, INRIA, February 2001. (number 4113).

[20] T. Sasaki and M. Sasaki. On Integer–to–Rational Conversion Algorithm. *SIGSAM Bulletin*, 26 :19–21, 1992.

[21] James Victor Uspensky. *Theory of Equations*. McGraw – Hill Co., New–York, 1948.

[22] A. J. H. Vincent. Sur la résolution des équations numériques. *Journal de Mathématiques Pures et Appliquées*, 1 :341–372, 1836.

[23] Paul Shyh-Horng Wang. A p–adic algorithm for univariate partial fractions. In *proceedings of ACM SYMSAC'81*, pages 212–217, 1981.