# Calcul formel : mode d'emploi

## Exemples en Maple

Philippe Dumas
Claude Gomez
Bruno Salvy
Paul Zimmermann

# Avant-Propos

Le calcul formel est désormais accessible à un grand nombre d'ingénieurs, de chercheurs ou d'étudiants, et est appelé à prendre une place de plus en plus grande dans les universités, les laboratoires ou les bureaux d'étude. Les systèmes de calcul formel sont en effet déjà disponibles sur les ordinateurs portables et certains le sont sur des ordinateurs de poche. Ce livre s'adresse à des utilisateurs, débutants ou non, qui ne sont pas forcément spécialistes du domaine, mais souhaitent exploiter au mieux les possibilités de leur système.

De nombreux calculs, trop longs pour être menés à bien à la main, peuvent être effectués avec sûreté en quelques secondes par une utilisation judicieuse d'un système de calcul formel, mais il faut savoir se retrouver parmi les centaines de fonctions présentées dans les manuels de référence. Pour cela, il faut dépasser l'approche "boîte noire" où on pose une question au système, et on attend qu'il veuille bien renvoyer une réponse, sans comprendre pourquoi il échoue ou parvient à un résultat. Il est important de connaître quelques opérations sur les objets de base (entiers, polynômes, fractions rationnelles, séries), et de savoir quelles *classes de problèmes* sait résoudre un système (quelles équations différentielles, quels calculs de primitives, quels types de systèmes d'équations,...).

Il n'est pas nécessaire d'être informaticien pour lire ce livre, mais la pratique d'un langage de programmation facilitera la lecture de certains passages. D'un point de vue mathématique, ce livre peut être abordé avec le niveau des classes préparatoires ou du premier cycle des universités ; certains exercices sont pris parmi les questions posées aux concours d'entrée aux grandes écoles.

Bien que l'essentiel de ce livre s'applique à tous les systèmes de calcul formel, il était important d'en choisir un pour traiter des exemples concrets et montrer les problèmes que ce traitement pose. Nous avons choisi le système Maple, et tous les exemples ont été réalisés en version V.3. La plupart d'entre eux peuvent être reproduits tels quels dans les autres versions de Maple.

Tous les calculs ont été réalisés sur une station de travail Dec Alpha 3000-400 sous Unix avec le système de gestion de fenêtres X Window. Les temps de calcul, lorsqu'ils sont indiqués, correspondent donc à cette machine. Mais tous les exemples peuvent être réalisés sur n'importe quel ordinateur disposant d'une mémoire suffisante.

Dans la première partie du livre, les principales fonctionnalités du système sont présentées (chap. I), ainsi que l'utilisation de son langage de programmation (chap. II).

En seconde partie, on aborde successivement les principaux champs d'application du calcul formel. D'abord le tracé de courbes (chap. III), puis les objets de base que sont les entiers (chap. IV), les structures représentables par un nombre fini d'objets de base : l'algèbre linéaire (chap. V), les polynômes et les fractions rationnelles (chap. VI), suivis par des objets plus symboliques : les suites (chap. VII), les séries et les développements asymptotiques (chap. VIII), les fonctions dans les chapitres sur l'intégration (IX), le calcul différentiel (X) et les calculs en probabilités (XI). Enfin on conclut sur le lien entre calcul formel et calcul numérique, en particulier en abordant le problème de l'interface avec les langages efficaces comme Fortran ou C (chap. XII).

Un livre sur le calcul formel est en même temps un livre sur le calcul. Plusieurs des méthodes présentées dans cet ouvrage ne sont pas enseignées en premier cycle, mais nous illustrons systématiquement ces méthodes par des exemples.

Si l'on utilise un système de calcul formel sans avoir une idée précise de ce que signifie la question qu'on lui pose, il est fort possible que la réponse du système soit fausse. L'utilisation optimale de ce livre pour l'étudiant est donc en conjonction avec un cours de mathématiques autour duquel il pourra mener des expérimentations faciles sans passer par des calculs pénibles, et ainsi disposer de plus de temps pour comprendre les concepts. Pour le chercheur ou l'ingénieur, nous espérons leur donner des balises pour formuler leurs problèmes, de façon à ce que le système puisse aider à les résoudre.

# Contents

# Introduction

En 1858, Charles Delaunay s'adressa en ces termes à l'Académie des Sciences : "J'ai l'honneur de faire part à l'Académie de l'achèvement des calculs que j'ai entrepris il y a plus de douze ans..." Deux ans plus tard, il ajouta : "J'ai l'honneur de présenter à l'Académie le tome XXVIII de ses Mémoires, formant le premier volume de ma Théorie du mouvement de la Lune". Deux tomes des Mémoires de l'Académie des Sciences seront consacrés à la théorie de Delaunay, le tome XXVIII publié en 1860 (883 pages) et le tome XXIX publié en 1867 (931 pages).

*La Théorie du Mouvement de la Lune.* L'étude du mouvement de la Lune se ramène au calcul de ce qu'on appelle la *fonction perturbatrice*. Cette fonction prend en compte l'action de la Terre et les perturbations dues au Soleil. Pour obtenir une approximation précise du mouvement de la Lune, Delaunay calcule un développement de la fonction perturbatrice par rapport à quatre petites quantités : les excentricités des orbites de la Lune et du Soleil, le sinus du demi-angle entre les plans d'orbite de la Lune et du Soleil, et le rapport des distances moyennes à la Terre de la Lune et du Soleil.

Les opérations nécessaires pour mener à bien ce calcul sont la recherche de développements limités à une ou plusieurs variables, la dérivation, l'intégration et la linéarisation d'expressions trigonométriques. Aujourd'hui, à l'aide d'un système sachant effectuer ces opérations, on obtient en quelques minutes le développement de la fonction perturbatrice au même ordre que Delaunay (voir l'exercice p. 205).

Sans pour autant discréditer le travail gigantesque de Delaunay (une seule erreur a été trouvée dans sa formule de cent trente-huit pages de la fonction perturbatrice), cet exemple donne un aperçu des nouveaux outils de calcul scientifique disponibles aujourd'hui sur ordinateur.

*Qu'est-ce que le calcul formel ?* Selon le contexte, l'expression *calcul formel* — on dit aussi *calcul symbolique*, plus rarement *calcul mathématique assisté par ordinateur* — a des sens différents. Nous en distinguons trois. Quand on dit "Delaunay a fait *un* calcul formel", on veut dire par là un calcul symbolique, par opposition à un calcul purement numérique. Dans la phrase "*le* calcul formel est en pleine évolution", on désigne la discipline recouvrant les opérations symboliques sur ordinateur. Enfin, quand on parle

d'un "système de calcul formel", cela signifie un logiciel permettant de faire des calculs mathématiques exacts, c'est-à-dire à peu près ce que l'on apprend en classe préparatoire aux grandes écoles scientifiques ou dans le premier cycle des universités, ce qui ne l'empêche pas de savoir faire aussi du calcul numérique et des tracés graphiques.

*Histoire du calcul formel.* Les premiers calculs symboliques sur ordinateur ont été réalisés il y a plus de quarante ans. Il s'agissait alors d'opérations spécifiques, comme le calcul de dérivées de fonctions. Les tout premiers systèmes étaient en général spécialisés et écrits par une ou deux personnes (Alpak par Brown en 1964, Formac par Bond et Tobey en 1964). Ces systèmes ont disparu depuis, faute de moyens humains et de développement. Sont apparus ensuite Reduce en 1968, Matlab en 1968 qui a donné Macsyma en 1970, et Scratchpad, développé par IBM dès le milieu des années soixante, qui est devenu Scratchpad II en 1975, pour n'être diffusé officiellement qu'en 1991 sous le nom d'Axiom. Ces trois systèmes (Reduce, Macsyma et Scratchpad) ont été écrits en Lisp. On a longtemps pensé que ce langage était préférable pour développer un système de calcul formel, jusqu'à l'apparition vers le milieu des années 1970 du langage C, dans lequel ont été écrits Maple (1980) et Mathematica (1988), successeur de SMP (1982).

*Le calcul formel aujourd'hui.* Il a acquis une notoriété considérable depuis 1988 avec l'arrivée de Mathematica, dont le concepteur, Stephen Wolfram, a mené une campagne de publicité impressionnante partout dans le monde. Cette publicité a fait mieux connaître le calcul formel dans le milieu industriel.

Les principaux systèmes de calcul formel utilisés actuellement sont Axiom, Macsyma, Maple, Mathematica et Reduce. Tous les cinq sont des systèmes généraux, c'est-à-dire qu'ils savent manipuler des nombres en précision arbitraire, factoriser ou développer des polynômes et fractions à nombre quelconque de variables, dériver — et intégrer lorsque c'est possible — des expressions construites à l'aide de fonctions élémentaires, résoudre des équations, différentielles ou non, de façon exacte ou à défaut numérique, effectuer des développements limités à un ordre quelconque, manipuler des matrices à coefficients symboliques, tracer des graphiques en deux ou trois dimensions. Ces systèmes évoluent sans cesse, au rythme d'une nouvelle version tous les ans environ.

Il existe aussi des logiciels spécialisés pour certains calculs symboliques. Ces logiciels ne fournissent pas tous les outils que propose un système général, mais ils disposent de fonctionnalités spécifiques à un domaine qu'ils sont souvent les seuls à offrir. En outre, dans leur domaine, ces logiciels sont généralement plus efficaces que les logiciels généraux. C'est le cas de Pari et Kant en théorie des nombres, de Cayley (devenu Magma) et Gap en théorie des groupes, de Macaulay pour les manipulations d'idéaux, de Gb pour les calculs de bases de Gröbner.

*Note sur les exemples.* Nous avons choisi Maple pour illustrer ce livre car, parmi les systèmes actuels, il nous semble être le seul à être à la fois d'accès facile, très diffusé (aussi bien par le nombre d'utilisateurs que par la diversité des machines le supportant), disposant d'une bibliothèque suffisamment riche et ouverte (on peut lire les sources de la plupart des fonctions), et aisément extensible.

Pour faire ressortir les commandes données à Maple dans les exemples, celles-ci sont affichées dans une police de caractères spéciale. En revanche, les résultats renvoyés sont affichés sensiblement comme Maple le fait lui-même :

```
sum(1/n^2,n=1..infinity);
```

$$\frac{\pi^2}{6}$$

Pour économiser l'espace, il nous arrive de regrouper plusieurs instructions sur une seule ligne, comme

```
assume(R1>0); assume(R2>0);
```

ou bien de condenser plusieurs instructions en une seule :

```
s1:=normal(subs(coordsI,y/sqrt(x^2+y^2)));
```

Le lecteur pourra décomposer les différentes commandes en faisant

```
tmp1:=y/sqrt(x^2+y^2);
tmp2:=subs(coordsI,tmp1);
s1:=normal(tmp2);
```

ce qui lui permettra de mieux suivre les étapes du calcul.

*Remerciement.* Philippe Dumas a relu patiemment plusieurs versions de chacun des chapitres de ce livre. Ses commentaires, tant sur le plan de la correction que de la pédagogie, ont grandement contribué à la lisibilité de ce livre. Nous l'en remercions vivement, ainsi que J.-C. Fort, pour ses commentaires sur une première version du chapitre XI.

# Première partie

# Système de calcul formel : mode d'emploi

CHAPTER I

# Prise en main du système

Un système de calcul formel peut être employé comme une calculatrice de poche. D'abord, comme un aide-mémoire, le système retrouve des formules que l'utilisateur peut avoir oubliées. Ensuite, et c'est l'emploi le plus important, le système effectue des calculs longs et fastidieux que l'utilisateur saurait faire lui-même. Le logiciel apporte alors vitesse et sûreté. Enfin, et c'est un usage fréquent mais dangereux si les résultats sont acceptés aveuglément, certaines fonctionnalités des systèmes sont employées comme "boîtes noires" pour effectuer des calculs que l'utilisateur ne saurait pas mener à bien.

Nous détaillons tout d'abord un exemple à travers lequel nous prenons un premier contact avec le système Maple et nous introduisons les notions élémentaires. Ensuite nous donnons un panorama des principales classes d'expressions manipulées par les systèmes, ce qui permet d'aborder les manipulations de base d'un système de calcul formel et le problème important de la simplification des expressions.

## 1. Une session détaillée

Le problème consiste à trouver l'aire de l'intersection de deux cercles, le centre de l'un étant situé sur l'autre cercle. Le traitement de cet exemple à l'aide de Maple présente la syntaxe de ce système ; les principes utilisés, comme pour la plupart des exemples de ce livre, sont cependant valables pour tous les systèmes de calcul formel. À titre de comparaison, ce même exemple est traité à l'aide de Mathematica en annexe A.

La première partie de l'étude consiste à fixer les notations. Le plan est muni d'un repère orthonormé d'axes $Ox$ et $Oy$. Les deux cercles $(C_1)$ et $(C_2)$ ont pour rayons respectifs $R_1$ et $R_2$. Sans restreindre la généralité du problème, le point $O$ est pris pour centre du cercle $(C_1)$ et $(C_2)$ est centré en $O_2$ sur l'axe $Ox$ (voir fig. 1 p. 8). Les équations des deux cercles sont donc :

$$\begin{aligned} (C_1): &\quad x^2 + y^2 = R_1^2, \\ (C_2): &\quad (x - R_1)^2 + y^2 = R_2^2. \end{aligned}$$

Si $I$ et $J$ sont les points d'intersection des deux cercles, l'aire cherchée est égale à la somme des deux différences d'aires suivantes :

- l'aire du secteur $OIJ$ du cercle $(C_1)$ moins l'aire du triangle $OIJ$,

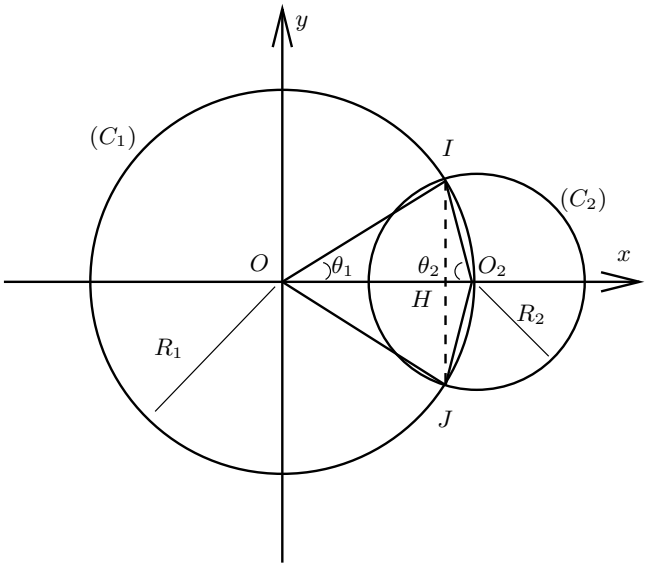



FIGURE 1 Les deux cercles $(C_1)$ et $(C_2)$.

- l'aire du secteur $O_2IJ$ du cercle $(C_2)$ moins l'aire du triangle $O_2IJ$.

Le problème est conceptuellement très simple, mais ces calculs d'aire sont assez lourds. Ces deux conditions en font un candidat idéal pour l'utilisation du calcul formel.

À ce stade, le lecteur peut lancer Maple pour suivre pas à pas les étapes de cette session.

Lorsque Maple est activé, apparaît une fenêtre appelée *worksheet* qui se présente de façon différente selon le type d'ordinateur utilisé. La figure 2 montre l'aspect obtenu sur une station de travail avec le gestionnaire de fenêtres X Windows.

Le principe de *worksheet* est le même pour tous les systèmes. L'utilisateur entre ses expressions dans la syntaxe Maple et le résultat est affiché après chaque calcul. L'affichage est présenté sous une forme *haute résolution* qui ressemble à la typographie mathématique. C'est le cas en particulier pour les lettres grecques (comme `alpha` dans la figure 2) et quelques autres symboles (racines carrées, sommes et intégrales).

Chaque *worksheet* comporte trois types de zones : les zones d'entrée (ou *input*), les zones d'affichage de résultat (ou *output*) et les zones de commentaire ou de dessin (*text*). Les zones d'entrée et de texte sont modifiables avec les commandes habituelles d'édition. En particulier, la souris sert à se déplacer, copier ou insérer du texte. La version actuelle ne permet pas la saisie de caractères accentués.

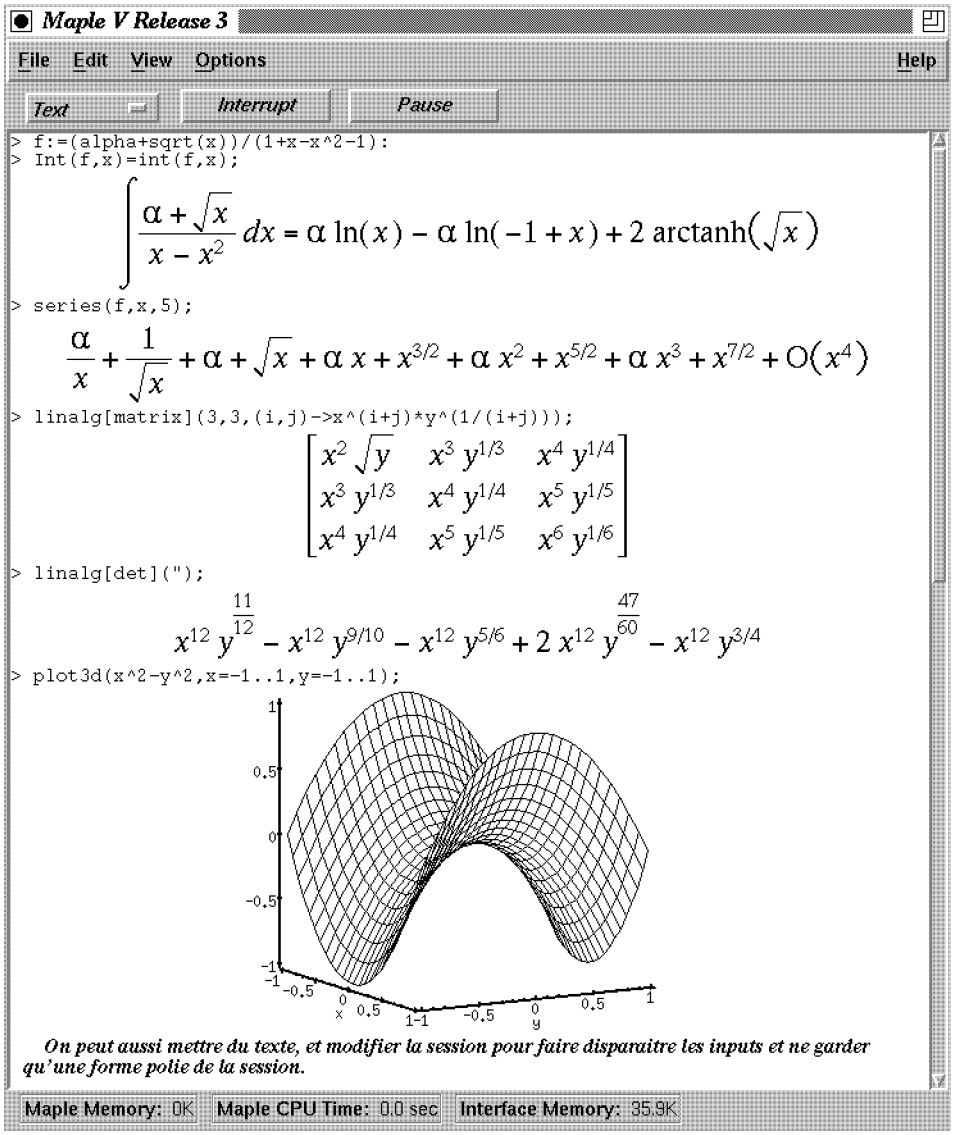



FIGURE 2 Fenêtre Maple sous X Windows.

Nous allons maintenant saisir les équations définissant les cercles $(C_1)$ et $(C_2)$.

Toute commande Maple doit être terminée par un point virgule “;” ou par deux points “:”. C’est impératif car c’est ce qui indique à Maple que la commande est terminée, “;” signifiant que le résultat doit être affiché et “:” qu’il ne doit pas l’être, par exemple pour des calculs intermédiaires dont

l'affichage serait très long. En revanche, les passages à la ligne au milieu d'expressions n'ont pas d'importance et servent à les présenter plus clairement.

Voici donc comment saisir les équations des deux cercles :

```
eq1 := x^2+y^2=R1^2;
```

$$eq1 := x^2 + y^2 = R1^2$$

```
eq2 := (x-R1)^2+y^2=R2^2;
```

$$eq2 := (x - R1)^2 + y^2 = R2^2$$

Ces commandes font apparaître deux types d'égalité. L'égalité mathématique est représentée par le signe `=` et sert à manipuler des équations. L'affectation, représentée par le signe `:=`, est une opération informatique qui permet de nommer des expressions. Ainsi, l'équation du cercle $(C_1)$ est désormais connue du système sous le nom `eq1`, tandis que `eq2` est la variable désignant l'équation de $(C_2)$.

L'usage de symboles est la différence essentielle entre les systèmes de calcul formel et les langages de programmation traditionnels comme C, Fortran, Pascal, Lisp,... Dans ces derniers, les opérations élémentaires s'effectuent sur des valeurs numériques. Dans les systèmes de calcul formel, ces opérations s'appliquent également à des symboles, comme dans notre exemple $x$, $y$, $R1$ et $R2$. On peut aussi noter à ce stade que Maple distingue majuscules et minuscules.

Une fois les équations saisies, l'étape suivante consiste à résoudre le système formé par ces deux équations pour trouver les coordonnées des points d'intersection.

Pour trouver comment effectuer cette résolution, nous utilisons l'aide en ligne. Maple est entièrement auto-documenté. Un menu d'aide permet d'avoir accès à un *help browser*. Là se trouvent décrits tous les objets et fonctions de Maple. C'est le moyen le plus efficace pour trouver le nom de la fonction réalisant un calcul donné (c'est en général le nom anglais de l'opération mathématique). Une fois ce nom trouvé, ici *solve*, la commande `help(solve)`, ou sa forme abrégée `?solve`, fournit des informations sur la fonction (syntaxe, arguments,...).

Le début de l'aide en ligne décrit brièvement le rôle et l'usage de la fonction :

```
> ?solve

FUNCTION: solve - Solve Equations

CALLING SEQUENCE:
   solve(eqn, var)
   solve(eqns, vars)

PARAMETERS:
   eqn  - an equation or inequality
   eqns - a set of equations or inequalities
   var  - (optional) a name (unknown to solve for)
   vars - (optional) a set of names (unknowns to solve for)
```

Ensuite une partie appelée *synopsis* décrit complètement la fonction. Une partie appelée *examples* donne des exemples simples d'utilisation, qu'il est possible d'exécuter soi-même. Enfin, une partie très importante appelée *see also* donne le nom de fonctions ayant un rapport avec celle dont on a demandé la documentation. Cela permet souvent de trouver de proche en proche le nom de la fonction désirée.

L'accès facile à cette aide en ligne est capital pour un système de calcul formel. En effet, il est impossible à un utilisateur de connaître le nom et encore moins la syntaxe du millier de fonctions que contient le système. L'aide en ligne contient tout le manuel de référence du système, et grâce à la souris on s'y déplace bien plus facilement que dans la version papier.

Pour revenir à notre problème, nous utilisons la fonction `solve` avec la syntaxe `solve(eqns,vars)` :

```
solve({eq1,eq2},{x,y});
```

$$\left\{x = \frac{1}{2}\frac{2R1^2 - R2^2}{R1}, y = \frac{1}{2}\frac{R2\sqrt{4R1^2 - R2^2}}{R1}\right\},$$
$$\left\{x = \frac{1}{2}\frac{2R1^2 - R2^2}{R1}, y = -\frac{1}{2}\frac{R2\sqrt{4R1^2 - R2^2}}{R1}\right\}$$

Les expressions entre accolades $\{\cdots\}$ représentent des ensembles. Nous résolvons l'ensemble (donc le système) `{eq1,eq2}` d'équations, par rapport à l'ensemble `{x,y}` d'inconnues. Le résultat est donné sous la forme d'un objet Maple appelé en anglais *expression sequence* et que nous appellerons *suite d'expressions*. Une suite d'expressions est une suite ordonnée d'expressions séparées par des virgules. Elle peut comprendre zéro élément (être vide) ; dans ce cas elle est représentée en Maple par le symbole `NULL`.

Il faut faire attention à ne pas confondre une suite d'expressions avec un ensemble (entre accolades) ou une liste (entre crochets, nous n'en avons pas encore rencontré). Ces trois types d'objets sont décrits au §2.5 et le tableau 9 p. 29 résume leurs propriétés.

La fonction `solve` donne deux solutions correspondant aux deux points $I$ et $J$ d'intersection des cercles. Chacune est donnée sous la forme d'un ensemble d'équations donnant les valeurs de $x$ et de $y$. La première solution, avec $y > 0$, correspond au point $I$ situé au-dessus de l'axe des abscisses.

Nous n'avons pas donné de nom au résultat de `solve`. Pour s'y référer, nous utilisons donc le caractère spécial `"` qui par définition a pour valeur le dernier résultat calculé par Maple. On peut aussi utiliser `""` et `"""` pour faire référence respectivement à l'avant-dernier et à l'antépénultième résultat calculé par Maple. L'opérateur de sélection `[···]` permet d'extraire des éléments d'une liste, d'un ensemble ou d'une suite d'expressions. Donc `"[1]` a pour valeur l'ensemble d'équations donnant les coordonnées du point $I$.

```
coordsI:="[1]:
```

L'aire des triangles se déduit des valeurs de $\sin\theta_1$, $\cos\theta_1$, $\sin\theta_2$ et $\cos\theta_2$ (fig. 1 p. 8), que nous nommerons respectivement `s1`, `c1`, `s2` et `c2`. Par

exemple, $\sin\theta_1$ est donné par

$$\sin\theta_1 = \frac{y}{\sqrt{x^2+y^2}}. \tag{1}$$

Pour obtenir cette valeur en Maple, deux approches sont possibles. La première consiste à affecter les coordonnées de $I$ aux variables $x$ et $y$. Ceci est permis en Maple, où le nom $x$ représente à la fois un symbole mathématique et une variable informatique. La commande `assign` (appliquée par exemple à `coordsI`) effectue cette affectation. La seconde approche, que nous préférons, consiste à substituer dans le membre droit de l'équation (1) les symboles $x$ et $y$ par les coordonnées de $I$. Cette opération est effectuée par la commande `subs`.

```
s1:=normal(subs(coordsI,y/sqrt(x^2+y^2)));
```

$$s1 := \frac{1}{2}\frac{R2\sqrt{4R1^2-R2^2}}{R1\sqrt{R1^2}}$$

```
s2:=normal(subs(coordsI,y/sqrt((R1-x)^2+y^2)));
```

$$s2 := \frac{1}{2}\frac{R2\sqrt{4R1^2-R2^2}}{R1\sqrt{R2^2}}$$

```
c1:=normal(subs(coordsI,x/sqrt(x^2+y^2)));
```

$$c1 := \frac{1}{2}\frac{2R1^2-R2^2}{R1\sqrt{R1^2}}$$

```
c2:=normal(subs(coordsI,(R1-x)/sqrt((R1-x)^2+y^2)));
```

$$c2 := \frac{1}{2}\frac{R2^2}{R1\sqrt{R2^2}}$$

Pour obtenir un résultat simplifié, nous avons employé la commande `normal` qui réduit au même dénominateur les fractions rationnelles, et divise numérateur et dénominateur par leur pgcd. À titre de comparaison, voici le résultat brut (non simplifié) donné par Maple pour le calcul de $\cos\theta_2$ :

```
subs(coordsI,(R1-x)/sqrt((R1-x)^2+y^2));
```

$$2\frac{-\frac{1}{2}\frac{2R1^2-R2^2}{R1}+R1}{\sqrt{4\left(-\frac{1}{2}\frac{2R1^2-R2^2}{R1}+R1\right)^2+\frac{R2^2(4R1^2-R2^2)}{R1^2}}}$$

Les fonctions de simplification sont décrites en détail dans la seconde partie de ce chapitre.

L'aire cherchée s'obtient alors en faisant la somme des différences des secteurs circulaires et des aires des triangles. Les aires des secteurs circulaires s'obtiennent facilement à partir des angles $\theta_1$ et $\theta_2$. Pour obtenir ceux-ci à partir des valeurs que nous avons déjà, il suffit d'appliquer la commande `arccos` à leurs cosinus. Maple dispose ainsi de toutes les fonctions mathématiques usuelles (logarithme, exponentielle, fonctions trigonométriques, fonctions trigonométriques hyperboliques...).

La multiplication est définie à l'aide de `*` qui est impératif.

```
A:=normal(R1^2*(arccos(c1)-s1*c1)+R2^2*(arccos(c2)-s2*c2));
```

$$A := \arccos\left(\frac{1}{2}\frac{2R1^2 - R2^2}{R1\sqrt{R1^2}}\right) R1^2 - \frac{1}{2}R2\sqrt{4R1^2 - R2^2} + R2^2 \arccos\left(\frac{1}{2}\frac{R2^2}{R1\sqrt{R2^2}}\right)$$

Cette formule n'est pas totalement satisfaisante : certaines simplifications manquent, en particulier

$$\sqrt{R_1^2} \mapsto R1 \qquad \text{et} \qquad \sqrt{R_2^2} \mapsto R_2.$$

Cette transformation est valide puisque $R_1$ et $R_2$ sont des réels positifs, mais le système de calcul formel ne le sait pas, en particulier il pourrait s'agir de nombres complexes. Plusieurs techniques permettent de résoudre ce problème et la discussion complète sera faite au §2. Une solution simple consiste à simplifier l'expression par la commande `simplify` avec comme arguments `sqrt` (pour simplifier les racines carrées) et `symbolic` (pour forcer la transformation $\sqrt{x^2} \mapsto x$).

```
A:=simplify(A,sqrt,symbolic);
```

$$A := \arccos\left(\frac{1}{2}\frac{2R1^2 - R2^2}{R1^2}\right) R1^2 - \frac{1}{2}R2\sqrt{4R1^2 - R2^2} + R2^2 \arccos\left(\frac{1}{2}\frac{R2}{R1}\right)$$

Comme l'immense majorité des commandes Maple, la commande `simplify` ne modifie pas son argument. Elle calcule un résultat et le renvoie. Pour modifier `A`, il faut donc lui affecter le résultat de `simplify`.

Cette formule n'aurait sans doute pas été très difficile à obtenir à l'aide d'une feuille de papier et d'un stylo. Cependant le grand nombre de simplifications nécessaires à son obtention rend le risque d'erreur élevé. Le calcul formel apporte une sécurité supplémentaire : nous sommes sûrs que les calculs sont justes ! En revanche les expressions que nous avons données à Maple ne le sont peut-être pas. Une simple vérification s'avère utile.

Le calcul de l'aire que nous avons réalisé est valable pour $R_2$ variant entre 0 et $2R_1$. Pour $R_2 = 0$, l'aire de l'intersection vaut 0 et pour $R_2 = 2R_1$ elle vaut l'aire du cercle $(C_1)$ soit $\pi R_1^2$. Pour calculer la valeur de `A` lorsque $R2 = 0$, il suffit de substituer 0 à `R2` dans `A` par la commande `subs` que nous avons déjà vue. Nous illustrons ici l'autre méthode, moins pratique, qui consiste à utiliser le symbole `R2` comme un nom de variable, et à affecter 0 à cette variable.

```
R2:=0: A;
```

$$0$$

Le résultat est bien vérifié. Le premier calcul est terminé par "`:`", ce qui dispense de l'affichage du résultat.

L'affectation de 0 à `R2` a modifié la valeur de `A` qui vaut à présent 0. C'est un piège classique en Maple. Dès qu'une valeur est affectée à un nom, cette valeur est remplacée dans toutes les expressions où apparaît le nom, y compris dans les expressions définies *avant* l'affectation, comme `A` ici. En fait, la valeur de l'aire que nous avons calculée n'est pas perdue. Pour la retrouver il suffit d'annuler l'affectation du nom `R2`. Pour cela on donne comme valeur

à `R2` le symbole `R2`. Ce dernier s'obtient en entourant le nom d'apostrophes. Dans l'exemple suivant `a` vaut 2 mais `'a'` vaut `a`.

```
a:=2: a,'a';
```

$$2, a$$

Annulons donc l'affectation de `R2` :

```
R2:='R2': A;
```

$$\arccos\left(\frac{1}{2}\frac{2R1^2 - R2^2}{R1^2}\right) R1^2 - \frac{1}{2} R2\sqrt{4R1^2 - R2^2} + R2^2 \arccos\left(\frac{1}{2}\frac{R2}{R1}\right)$$

et `A` retrouve bien la valeur qui lui a été donnée.

Il apparaît clairement que ce procédé (affectation puis annulation de celle-ci) est bien lourd. Pour évaluer une expression pour diverses valeurs de ses paramètres, l'utilisation de la commande `subs` est préférable :

```
subs(R2=0,A);
```

$$\arccos(1) R1^2$$

Cet exemple illustre une particularité de la commande `subs`. Nous avons demandé à Maple de remplacer `R2` par 0 dans `A`. Le résultat est juste mais la fonction `subs` ne réalise que la substitution, sans évaluer les fonctions qui pourraient l'être dans le résultat. C'est un des rares cas où il est nécessaire d'utiliser la commande d'évaluation `eval` de Maple. Il faut le faire chaque fois qu'à l'issue d'une substitution, les fonctions présentes dans l'expression doivent être réévaluées. La bonne solution est finalement

```
eval(subs(R2=0,A));
```

$$0$$

De même on peut vérifier la valeur de l'aire pour $R_2$ valant $2R_1$ :

```
eval(subs(R2=2*R1,A));
```

$$\pi R1^2$$

Le résultat est bien celui attendu. Il fait apparaître la constante Maple `Pi` qui à la différence des programmes numériques n'est pas une approximation mais représente bien le nombre transcendant $\pi$.

Maple connaît d'autres constantes comme `E` qui représente la base $e$ des logarithmes népériens, ou la constante $\gamma$ d'Euler représentée par `gamma`. De même, le symbole `infinity` représente $+\infty$ et `-infinity` représente $-\infty$.

Pour conclure ce problème, nous étudions le rapport qui doit exister entre les rayons $R_2$ et $R_1$ pour que l'aire de l'intersection des deux cercles soit égale à la moitié de l'aire du cercle $(C_1)$. Si $K$ est ce rapport, nous commençons par remplacer dans la valeur de l'aire le rayon $R_2$ par $KR_1$ :

```
AA:=normal(subs(R2=K*R1,A));
```

$$AA := \pi R1^2 - R1^2 \arccos\left(-1 + \frac{1}{2}K^2\right) - \frac{1}{2} KR1\sqrt{4R1^2 - K^2 R1^2} + K^2 R1^2 \arccos\left(\frac{1}{2}K\right)$$

Il ne reste plus qu'à résoudre l'équation en $K$ exprimant que l'aire `AA` vaut $\pi R_1^2/2$.

```
solve(AA=Pi*R1^2/2,K);
```

Maple ne trouve aucune solution et retourne comme valeur la suite d'expressions vide `NULL` dont l'affichage est vide (essayer `NULL;`). En effet, il n'y a pas de solution analytique à ce problème. En revanche, il est possible de calculer une valeur numérique approchée de $K$. Dans ce cas c'est la fonction `fsolve` qui est utilisée. Cette fonction requiert une équation en une seule variable. Il nous faut donc diviser `AA` par $R_1^2$ et simplifier.

```
simplify(AA/R1^2,sqrt,symbolic);
```

$$\pi - \arccos\left(-1 + \frac{K^2}{2}\right) - \frac{K\sqrt{4-K^2}}{2} + K^2 \arccos\left(\frac{K}{2}\right)$$

```
fsolve("=Pi/2,K);
```

1.158728473

La précision par défaut est de 10 chiffres ; pour la modifier il suffit d'attribuer à la variable `Digits` le nombre de chiffres significatifs souhaités.

Pour finir, nous traçons la courbe (fig. 3 p. 15) donnant le rapport de l'aire de l'intersection et de l'aire de $(C_1)$ en fonction de $K$. La fonction à utiliser est `plot` dont la syntaxe est très simple :

```
plot(AA/(Pi*R1^2),K=0..2);
```

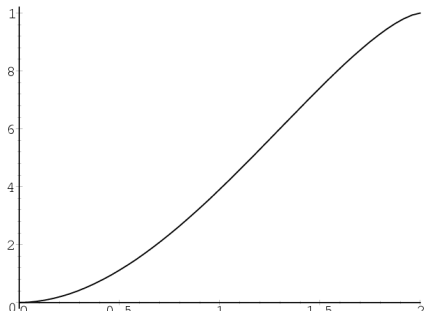



FIGURE 3 Rapport des aires en fonction de $K$.

En fait, nous venons de résoudre le problème suivant :

> Un paysan possède un pré circulaire de rayon égal à 100 mètres. Il attache sa chèvre à un pieu planté sur la circonférence du pré. Quelle doit être la longueur de la corde reliant la chèvre au pieu de façon que la chèvre puisse brouter exactement la moitié de la surface du pré ?

La réponse est donc 115,87 mètres.

## 2. Classes d'expressions et simplification

L'exemple traité en détail dans les pages précédentes montre que la résolution d'un problème passe par l'emploi de nombreuses fonctions : `solve`, `subs`, `normal`, `simplify`, `eval`, `fsolve`, `plot`. Outre la maîtrise de l'aide en ligne, il faut apprendre à raisonner en termes de classes d'expressions. Chaque fonction Maple s'applique à (et produit) une classe bien définie d'expressions. Reconnaître qu'une expression appartient à telle ou telle classe permet du même coup de savoir quelles fonctions on peut lui appliquer.

Un problème pour lequel cette reconnaissance est essentielle est celui de la simplification d'expressions. C'est autour de ce problème que sont définies les principales classes d'expressions des systèmes de calcul formel. En effet, dès qu'il est possible de déterminer si une expression appartenant à une classe est nulle ou non, il est possible d'effectuer des divisions dans cette classe. Autrement, tous les calculs qui demandent une division deviennent hasardeux. Dans les classes les plus simples, il existe une *forme normale*. Sous cette forme, deux expressions représentent le même objet mathématique si et seulement si elles sont identiques. Cependant, la représentation idéale n'est pas toujours la forme normale. Dans le cas des polynômes par exemple, la représentation développée est une forme normale, mais la représentation factorisée permet des calculs de pgcd bien plus rapides. Ce genre d'exemple amène les systèmes de calcul formel à un compromis. Un certain nombre de simplifications basiques, comme la réduction des rationnels ou la multiplication par zéro, sont effectuées automatiquement ; les autres récritures sont laissées à l'initiative de l'utilisateur auquel des commandes spécialisées sont proposées.

En Maple les principales fonctions permettant de récrire des expressions sont `normal`, `expand`, `combine`, `collect` et `simplify`. Pour bien utiliser ces fonctions, il faut savoir quel type de transformations elles effectuent et à quelle classe d'expressions ces transformations s'appliquent. Ainsi, l'usage aveugle de la fonction `simplify` peut conduire à des résultats faux. Un second argument de `simplify` permet néanmoins de préciser la simplification à effectuer. *Toute utilisation de* `simplify` *sans ce second argument est très dangereuse.*

Dans cette section, nous allons passer en revue les principales classes d'expressions d'un système de calcul formel et les fonctions de manipulation correspondantes. Nous insisterons sur les fonctions de récriture, c'est-à-dire celles qui modifient la forme d'une expression sans changer sa signification mathématique. Un premier aperçu est donné par le tableau 1. La plupart de ces classes d'expressions seront étudiées plus en détail dans la seconde partie de ce livre.

### 2.1. Classes élémentaires.

Les classes élémentaires sont formées d'expressions sans variable, c'est-à-dire de constantes : entiers, rationnels, nombres flottants, booléens, résidus modulo $p$ et nombres $p$-adiques.

2.1.1. *Entiers*

Dans un système de calcul formel les opérations sur des nombres entiers ou rationnels sont *exactes*.

Exemple 1. Un calcul typique d'entier est celui de factorielle 100.

```
100!;
```

$$933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000$$

De nombreuses fonctions s'appliquent aux entiers. Une sélection des plus importantes sera présentée au chapitre IV.

Exemple 2. Fermat avait conjecturé que tous les nombres de la forme $2^{2^n}+1$ étaient premiers. Voici le premier exemple qui invalide sa conjecture :

```
ifactor(2^(2^5)+1);
```

$$(641)(6700417)$$

Du point de vue de la simplification, tous les entiers sont représentés en base dix (ou deux selon les systèmes), ce qui constitue une forme normale. L'égalité d'entiers est donc facile à tester (en Maple, le test d'égalité syntaxique se fait en temps constant, indépendamment de la taille des objets). Toute opération sur des entiers est immédiatement effectuée ; par exemple, $2^2$ n'est pas représentable en Maple, il est immédiatement transformé en 4. Cela signifie aussi qu'un nombre factorisé ne peut pas être représenté comme un entier, puisqu'alors il serait immédiatement développé. Dans l'exemple précédent, le résultat est en réalité un produit de fonctions.

Table 1 Principaux simplificateurs.

| Classe d'expressions | Fonction |
|---|---|
| entiers | simplification automatique |
| rationnels | simplification automatique |
| flottants | `evalf` |
| booléens | `evalb` |
| résidus mod $p$ | `mod` |
| nombres $p$-adiques | `padic[evalp]` |
| matrices | `evalm` |
| fractions rationnelles | `normal` |
| développements limités | `series` |
| nombres algébriques | `evala` |
| racines carrées | `rationalize` |
| nombres complexes | `evalc` |
| fonction $f$ | `simplify(...,f)` |

2.1.2. *Rationnels*

La propriété de forme normale s'étend aux nombres rationnels. Non seulement les additions, multiplications et quotients sont immédiatement exécutés, mais en plus les fractions rationnelles sont toutes réduites.

Exemple 3. Dans cet exemple, les factorielles sont d'abord évaluées, puis le rationnel obtenu est simplifié :

```
99!/100!-1/50;
```

$$-\frac{1}{100}$$

2.1.3. *Flottants*

Les règles de simplification automatique sont moins systématiques pour les nombres approchés numériquement, appelés aussi nombres en virgule flottante, ou plus simplement *flottants*. Lorsqu'ils interviennent dans une somme, un produit ou un quotient faisant intervenir par ailleurs des rationnels, ils sont contagieux, c'est-à-dire que toute l'expression devient un nombre flottant.

Exemple 4.

```
72/53-5/3*2.7;
```

$$-3.141509435$$

Pour les autres expressions, la fonction de base pour ces calculs est `evalf` qui évalue numériquement une expression (tous les nombres sont transformés en flottants). Un argument optionnel permet de préciser le nombre de chiffres significatifs utilisés lors du calcul.

Exemple 5. Voici $\pi$ avec 50 chiffres significatifs

```
evalf(Pi,50);
```

$$3.1415926535897932384626433832795028841971693993751$$

La précision peut également être réglée par la variable globale `Digits`, qui vaut 10 par défaut.

Les flottants en Maple sont liés à leur précision : ainsi la valeur précédente est différente syntaxiquement de la valeur de $\pi$ calculée avec dix chiffres significatifs. Compte tenu de cette restriction, les flottants renvoyés par `evalf` sont sous forme normale. Le chapitre XII revient plus en détail sur l'usage des flottants en Maple.

2.1.4. *Booléens*

Les expressions booléennes forment aussi une classe élémentaire. Les deux formes normales sont `true` et `false`. Les autres expressions s'y réduisent par la commande `evalb`.

Exemple 6.

```
a:=0:b:=2:c:=3:
evalb(a=1 or (b=2 and c=3));
```

*true*

2.1.5. *Classes issues de l'arithmétique*

Les autres constantes formant une classe élémentaire munie d'une forme normale sont les résidus modulo $p$, avec pour fonction de réduction `mod`, et les nombres $p$-adiques, mis sous forme normale par la fonction `padic[evalp]`.

**2.2. Classes à forme normale.** À partir de constantes bien définies, des classes d'objets symboliques faisant intervenir des variables et admettant une forme normale peuvent être construites. Les plus importantes sont les matrices, les polynômes et fractions rationnelles, les développements limités et les nombres algébriques. Pour chacune de ces classes, nous indiquons les principales fonctions de récriture.

2.2.1. *Matrices*

La forme normale d'une matrice est obtenue lorsque tous ses coefficients sont eux-mêmes sous forme normale. Le système n'effectue aucune simplification automatique sur les matrices et il faut demander explicitement le calcul d'une somme, d'un produit ou d'une puissance de matrices. La fonction d'évaluation correspondante s'appelle `evalm`. Il faut utiliser l'opérateur `&*` pour les produits, car la multiplication commutative `*` simplifierait abusivement `a*b-b*a` en `0` et `a*b*a` en `a^2*b`, avant même que soient effectués les produits matriciels. De même il faut faire attention avec la puissance, car `A^0` est simplifié automatiquement en `1` au lieu d'être transformé en la matrice identité.

Exemple 7. La matrice identité s'obtient comme élément neutre de la multiplication sous la forme `&*()`. Nous utilisons la commande Maple `alias` qui permet de simplifier l'écriture en affichant `Id` au lieu de `&*()`.

```
alias(Id=&*()):
a:=array([[1,2,3],[2,4,8],[3,9,27]]);
```

$$a := \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 8 \\ 3 & 9 & 27 \end{bmatrix}$$

```
evalm( (a^2+ Id) &* a^(-1) );
```

$$\begin{bmatrix} -5 & 13/2 & 7/3 \\ 7 & 1 & 25/3 \\ 2 & 19/2 & 27 \end{bmatrix}$$

De nombreuses autres fonctions s'appliquent aux matrices. Elles sont détaillées au chapitre V.

2.2.2. *Polynômes et fractions rationnelles*

Les calculs sur les polynômes et les fractions rationnelles à une ou plusieurs indéterminées sont les opérations de base d'un système de calcul formel. Contrairement aux classes présentées jusqu'ici, il n'y a pas *une* bonne représentation des polynômes. Les fonctions permettant de *récrire* un polynôme sous

TABLE 2 Récritures de polynômes.

| Polynôme `p` | $zx^2+x^2-(x^2+y^2)(ax-2by)+zy^2+y^2$ |
|---|---|
| `collect(p,x)` | $(z+1+2by)x^2-y^2ax+2y^3b+zy^2+y^2-x^3a$ |
| `collect(p,[x,y])` | $(z+1+2by)x^2-y^2ax+2y^3b+(z+1)y^2-x^3a$ |
| `collect(p,[x,y],`<br>`  distributed)` | $(z+1)x^2+2byx^2-y^2ax+2y^3b+(z+1)y^2-x^3a$ |
| `expand(p)` | $zx^2+x^2-x^3a+2x^2by-y^2ax+2y^3b+zy^2+y^2$ |
| `factor(p)` | $(x^2+y^2)(-ax+z+1+2by)$ |

diverses formes sont résumées au tableau 2. Le tableau 3 résume celles concernant les fractions rationnelles. Des informations plus précises sur chacune de ces fonctions et sur les nombreuses autres opérations que peuvent subir polynômes et fractions rationnelles sont données au chapitre VI.

TABLE 3 Récritures de fractions rationnelles.

| Fraction `f` | Opération | Résultat |
|---|---|---|
| $\frac{x^3+3x^2+2x+yx^2+3yx+2y}{x^3+yx+2x^2+2y}$ | `normal(f)` | $\frac{x^2+yx+x+y}{x^2+y}$ |
| | `factor(f)` | $\frac{(x+1)(x+y)}{x^2+y}$ |
| $\frac{x^3+3x^2+2x+yx^2+3yx+2y}{x^3+x+2x^2+2}$ | `collect(f,y)` | $\frac{(x^2+3x+2)y}{x^3+x+2x^2+2}+\frac{x^3+3x^2+2x}{x^3+x+2x^2+2}$ |
| | `collect(f,y,normal)` | $\frac{(x+1)y}{x^2+1}+\frac{(x+1)x}{x^2+1}$ |
| $\frac{x^2+3yx+2}{x^2+1}$ | `expand(f)` | $\frac{x^2}{x^2+1}+\frac{3yx}{x^2+1}+\frac{2}{x^2+1}$ |

2.2.3. *Développements limités*

Comme les matrices et les flottants, les développements limités ont une forme normale, mais celle-ci n'est pas produite automatiquement. La commande de réduction est `series`. Comme pour `evalf`, l'ordre des développements est spécifié soit en donnant un argument supplémentaire à `series`, soit en modifiant la variable globale *Order*, qui vaut 6 par défaut.

EXEMPLE 8.

```
series(x/(2+x-x^2+O(x^3)),x);
```

$$\frac{1}{2}x-\frac{1}{4}x^2+\frac{3}{8}x^3+O(x^4)$$

```
s:=series(exp(sin(log(1+x))),x);
```

$$s:=1+x-\frac{1}{6}x^3+\frac{1}{12}x^4-\frac{1}{30}x^5+O(x^6)$$

```
series(s^2+x^3,x,4);
```

$$1+2x+x^2+\frac{2}{3}x^3+O(x^4)$$

Il est important de noter que si les coefficients des développements limités sont dans une classe d'expressions n'admettant pas de forme normale, alors les résultats renvoyés par `series` peuvent être faux.

EXEMPLE 9. Une des façons les plus simples de déguiser 0 consiste à l'écrire $\exp(1)\exp(-1) - 1$. Ceci conduit à une erreur de `series` :

```
f:=1/(z-z*exp(z)*exp(-1/z));
```

$$f := \frac{1}{z - ze^z e^{(-\frac{1}{z})}}$$

```
series(f,z=1,2);
```

$$\frac{1}{1-ee^{-1}} + \frac{1-3ee^{-1}}{(ee^{-1}-1)(1-ee^{-1})}(z-1) + O((z-1)^2)$$

Le premier terme est en réalité infini, `series` devrait repérer un pôle simple et produire $(1-z)^{-1}/2 + O(1)$, ce que l'on obtient en appliquant `series` à `combine(f,exp)` (tab. 5 p. 25).

2.2.4. *Nombres algébriques*

Un nombre algébrique est défini comme racine d'un polynôme. Lorsque le degré du polynôme est plus grand que 4, il n'est pas possible de le résoudre explicitement en général. Cependant, de nombreux calculs sur ses racines peuvent être menés à bien sans autre information que le polynôme lui-même. Ces calculs sont détaillés au chapitre VI.

Les nombres algébriques sont représentés en Maple par l'opérateur `RootOf` qui prend en argument le polynôme. Les fractions rationnelles en un nombre algébrique admettent une forme normale, calculée par `evala`.

EXEMPLE 10.

```
alias(alpha=RootOf(x^7+3*x^2+1,x)):
alpha^3/(alpha^8+3*alpha^2+1);
```

$$\frac{\alpha^3}{\alpha^8 + 3\alpha^2 + 1}$$

```
evala(");
```

$$-\frac{1}{5}\alpha^2 - \frac{34}{5} - \frac{11}{5}\alpha^6 - \frac{11}{5}\alpha^5 + \frac{4}{5}\alpha^4 + \frac{4}{5}\alpha^3 - \frac{34}{5}\alpha$$

Il faut noter que l'expression `RootOf(x^7+3*x^2+1,x)`, que nous avons fait afficher $\alpha$ pour une meilleure lisibilité à l'aide de la commande `alias`, représente l'une *quelconque* des sept racines du polynôme $x^7 + 3x^2 + 1$. Par contre, l'identité prouvée par `evala` n'est vraie que si les différentes occurrences du symbole $\alpha$ sont remplacées par la *même* racine.

2.2.5. *Racines carrées*

Pour simplifier des fractions rationnelles dont le dénominateur comprend des racines carrées, la méthode classique de multiplication par l'expression conjuguée est réalisée par la commande `rationalize`, qu'il faut charger au préalable par `readlib(rationalize)`.

```
rationalize(1/(1+sqrt(2)+sqrt(3)));
```

$$-\frac{1}{4}(-1-\sqrt{3}+\sqrt{2})(-1+\sqrt{3})$$

Pour obtenir une forme normale, il faut développer le résultat donné par la commande `rationalize` ; les facteurs obtenus au numérateur dépendent en effet de l'ordre d'élimination des racines carrées.

```
expand(");
```

$$\frac{1}{2}+\frac{1}{4}\sqrt{2}-\frac{1}{4}\sqrt{2}\sqrt{3}$$

Nous aurions pu obtenir le même résultat (mais plus laborieusement) à l'aide de `RootOf` et `evala`, en substituant `RootOf(x^2=n)` à `sqrt(n)`, puis en appliquant `evala`, et en effectuant la substitution inverse.

La commande `rationalize` accepte également des expressions contenant des racines imbriquées ou des variables :

```
rationalize(1/(sqrt(x-sqrt(y))+sqrt(z+t)));
```

$$-\frac{\left(-\sqrt{z+t}+\sqrt{x-\sqrt{y}}\right)\left(z+t-x-\sqrt{y}\right)}{z^2+2tz-2xz+t^2-2tx+x^2-y}$$

**2.3. Expressions complexes et simplification.** Les classes d'expressions présentées jusqu'ici partagent la propriété d'avoir une procédure de décision pour la nullité. C'est-à-dire que pour toutes ces classes un programme peut déterminer si une expression donnée est nulle ou non. Dans de nombreux cas, cette décision se fait par réduction à la forme normale : l'expression est nulle si et seulement si sa forme normale est le symbole 0.

Malheureusement, toutes les classes d'expressions n'admettent pas une forme normale. Pire encore, pour certaines classes il est impossible de prouver la nullité d'une expression en temps fini. Un exemple d'une telle classe est fourni par les expressions composées à partir des rationnels, des nombres $\pi$ et $\log 2$ et d'une variable, par utilisation répétée de l'addition, de la soustraction, du produit, de l'exponentielle et du sinus. Bien sûr, une utilisation répétée de `evalf` en augmentant la précision permet souvent de savoir si une expression particulière est nulle ou non ; mais Richardson a montré qu'il est impossible d'écrire un programme prenant en argument une expression de cette classe et donnant au bout d'un temps fini le résultat vrai si celle-ci est nulle, et faux sinon.

C'est dans ces classes que se pose avec le plus d'acuité le problème de la simplification. Sans forme normale, les systèmes ne peuvent que donner un certain nombre de fonctions de récriture avec lesquelles l'utilisateur doit jongler pour parvenir à un résultat. Pour y voir plus clair dans cette jungle, il faut là encore distinguer plusieurs sous-classes, savoir quelles fonctions s'appliquent et quelles transformations sont effectuées.

2.3.1. *Constantes*

Comme dit précédemment, les calculs se font avec des nombres entiers ou rationnels *exacts* et avec des constantes mathématiques *vraies* (qui ne sont pas des représentations flottantes).

Les constantes les plus simples sont les rationnels, le nombre $\pi$ noté `Pi`, la base $e$ des logarithmes népériens notée `E`, le nombre imaginaire $i$ noté `I` et la constante d'EULER $\gamma$ notée `gamma`.

Ces constantes sont relativement bien connues du système. Une exception est la constante `E`, peu utilisée par Maple, à laquelle il faut préférer `exp(1)`. Pour la classe simple des polynômes en $\pi$ et $e$, aucun algorithme de décision n'est connu : à ce jour on ignore s'il existe un tel polynôme non trivial qui vaille zéro.

En utilisation interactive, une bonne façon de traiter ces constantes dans des simplifications compliquées est de les remplacer toutes sauf $i$ par des variables et d'utiliser les procédures de forme normale des fractions rationnelles. Ceci revient à faire l'hypothèse que toutes ces constantes sont algébriquement indépendantes. Cette remarque se généralise à des constantes plus complexes comme $\ln 2$, $\exp(\pi + \log 3)$,... mais il faut alors être sûr que celles-ci ne sont pas trivialement dépendantes.

### 2.3.2. *Nombres complexes*

Les nombres complexes existent dans tous les systèmes de calcul formel. En Maple, on note `I` le nombre imaginaire $i$.

La fonction de base pour les calculs sur les nombres complexes est `evalc`. Elle met une expression sous la forme $a + ib$, où $a$ et $b$ sont réels. Comme la nullité n'est pas en général décidable, il en va de même de la réalité. Cependant, dès que $a$ et $b$ sont dans des classes à forme normale, `evalc` fournit une forme normale pour les complexes associés.

EXEMPLE 11. On peut par exemple calculer $\sqrt{i}$ :

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

$$(-1)^{1/4}$$

```
evalc(");
```

$$\frac{1}{2}\sqrt{2} + \frac{1}{2}I\sqrt{2}$$

Le résultat de ce calcul pose le problème de la détermination des racines. L'imaginaire $i$ a deux racines alors que `evalc` n'en donne qu'une. Dans le cas d'expressions plus compliquées, les choix multiples de racines carrées ou cubiques peuvent rendre la reconnaissance de 0 difficile, surtout si ces choix doivent être faits de manière cohérente dans l'expression.

Le même type de problème se pose avec toutes les fonctions multiformes, comme le logarithme ou les fonctions hypergéométriques. Le système fournit alors peu d'assistance pour les simplifications.

Par ailleurs, dans un calcul avec des expressions complexes, `evalc` suppose que les variables qui interviennent sont réelles. Tout ceci entraîne qu'il faut être très prudent avec la manipulation de nombres complexes.

Les autres commandes principales sont `Re`, `Im`, `abs` et `argument` donnant respectivement la partie réelle, la partie imaginaire, le module et l'argument.

```
z:=a+I*b: Re(z),Im(z),abs(z),argument(z);
```

$$\Re(a) - \Im(b), \Im(a) + \Re(b), |a + Ib|, \text{argument}(a + Ib)$$

Un appel à `evalc` simplifie en supposant $a$ et $b$ réels :

```
evalc(Re(z)),evalc(Im(z)),evalc(abs(z)),evalc(argument(z));
```

$$a, b, \sqrt{a^2 + b^2}, \arctan(b, a)$$

### 2.3.3. *Fonctions*

Nous avons vu lors de l'exemple de la section 1 que la plupart des fonctions mathématiques se retrouvent en Maple, en particulier les fonctions trigonométriques, le logarithme et l'exponentielle. La simplification de telles fonctions est cruciale. Le tableau 5 p. 25 décrit les commandes de base réalisant ces simplifications.

Tout ce qui concerne le classique tableau de variation de la fonction (calcul des dérivées, des asymptotes, des extremums, recherche des zéros et tracé de la courbe) peut être facilement réalisé à l'aide d'un système de calcul formel. Les principales opérations Maple qui s'appliquent à une fonction sont résumées au tableau 6 p. 25.

### 2.3.4. *Équations*

Un *leitmotiv* de ce livre est la manipulation d'objets définis par des équations, sans passer par la résolution de celles-ci.

Ainsi, une fonction définie par une équation différentielle linéaire et des conditions initiales est parfaitement précisée. L'ensemble des solutions d'équations différentielles linéaires est clos par addition et produit (entre autres) et forme ainsi une importante classe où l'on peut décider de la nullité. En revanche, si l'on résout une telle équation, la solution, privée de son équation de définition, tombe dans une classe plus grande où bien peu est décidable. Les chapitres de la seconde partie reviendront plus en détail sur ces considérations. Cependant, dans certains cas, surtout en utilisation interactive, il est utile de chercher une solution explicite, par exemple pour passer à une application numérique. Les principales fonctions de résolution sont résumées au tableau 4.

TABLE 4 Résolution d'équations.

| Commande | Usage |
|---|---|
| `fsolve` | solutions flottantes |
| `isolve` | solutions entières |
| `msolve` | solutions modulaires |
| `linsolve` | solutions d'équations linéaires |
| `dsolve` | solutions d'équations différentielles |
| `rsolve` | solutions de récurrences |
| `solve` | résolveur symbolique général |

TABLE 5 Simplifications des fonctions élémentaires.

| Commande | Résultat |
|---|---|
| `expand(···)` | $\sin(a+b) \mapsto \sin(a)\cos(b)+\cos(a)\sin(b)$ |
| | $\cos(a+b) \mapsto \cos(a)\cos(b)-\sin(a)\sin(b)$ |
| | idem pour les fonctions hyperboliques |
| | $e^{a+b} \mapsto e^a e^b$ |
| `combine(···,trig)` | $\cos(a)\cos(b) \mapsto \cos(a-b)/2+\cos(a+b)/2$ |
| | $\cos(a)\sin(b) \mapsto \sin(a+b)/2-\sin(a-b)/2$ |
| | $\sin(a)\sin(b) \mapsto \cos(a-b)/2-\cos(a+b)/2$ |
| | idem pour les fonctions hyperboliques |
| `combine(···,exp)` | $e^a e^b \mapsto e^{a+b}$ |
| | $(e^a)^b \mapsto e^{ab}$ |
| | $e^{a+n\ln b} \mapsto b^n e^a$ où $n$ est entier |
| `combine(···,ln)` | $n\ln b \mapsto \ln(b^n)$ où $n$ est entier |
| | $\ln a+\ln b \mapsto \ln(ab)$ |
| `simplify(···,trig)` | $\sin(x)^2+\cos(x)^2 \mapsto 1$ |
| | $\cosh(x)^2-\sinh(x)^2 \mapsto 1$ |
| `simplify(···,exp)` | $e^{a\ln b} \mapsto b^a$ |
| `simplify(···,ln)` | $\ln(b^a) \mapsto a\ln b$ |
| ou `expand(···)` | $\ln(ab) \mapsto \ln a+\ln b$ |
| `simplify(···,power)` | $x^a x^b \mapsto x^{a+b}$ |
| `simplify(···,power,symbolic)` | $(a^b)^c \mapsto a^{bc}$ |
| ou `combine(···,power)` | $\sqrt{a^2} \mapsto a$ |
| `convert(···,exp)` | $\cos(x) \mapsto (e^{ix}+e^{-ix})/2$ |
| | $\cosh(x) \mapsto (e^x+e^{-x})/2$ |
| `convert(···,trig)` | $e^{ix} \mapsto \cos(x)+i\sin(x)$ |
| | $e^x \mapsto \cosh(x)+\sinh(x)$ |
| `convert(···,ln)` | $\arccos(x) \mapsto -i\ln(x+i\sqrt{1-x^2})$ |
| | $\operatorname{arctanh}(x) \mapsto (\ln(1+x)-\ln(1-x))/2$ |

TABLE 6 Principales opérations sur les fonctions.

| Expression | Résultat |
|---|---|
| `f` | la fonction elle-même |
| `diff(f,x)` | dérivée par rapport à $x$ |
| `int(f,x)` | primitive par rapport à $x$ |
| `eval(subs(x=a,f))` | $f(a)$ |
| `limit(f,x=a)` | limite de $f(x)$ lorsque $x \to a$ |
| `series(f,x=a)` | développement limité en $x=a$ |
| `asympt(f,x)` | développement asymptotique lorsque $x \to \infty$ |
| `plot(f,x=a..b)` | la courbe $y=f(x)$ pour $x \in [a,b]$ |

TABLE 7 Calcul avec des formes inertes.

| Forme inerte | Fonctions qui s'y appliquent |
|---|---|
| `Int` | `diff`, `evalf`, `series` |
| `Sum` | `diff`, `evalf` |
| `Product` | `diff`, `evalf`, `mod` |
| `RootOf` | `evala`, `sum`, `factor`, `allvalues`, `testeq`, `solve` |
| | `evalf`, `product`, `diff`, `evalc`, `evalgf`, `series` |
| `DESol` | `diff`, `series` |
| `Diff` | `diff`, `D`, `liesymm`, `expand` |
| `Limit` | `evalf` |
| `Re`, `Im` | `evalc` |
| `Eigenvals` | `Svd`, `evalf` |

2.3.5. *Formes inertes en Maple*

Dans l'exemple 10 p. 21, nous avons représenté les racines du polynôme $x^7 + 3x^2 + 1$ à l'aide de `RootOf`. Cette fonction ne fait aucun calcul ; elle sert uniquement à représenter l'une des solutions d'une équation. De telles fonctions qui ne font rien sont appelées des fonctions *inertes*. Leur rôle est d'être reconnues par les fonctions du système qui effectuent des calculs.

`RootOf` n'est pas la seule fonction inerte de Maple. La fonction `DESol` représente les solutions d'une équation différentielle. La fonction `Int` représente une intégrale que le système ne cherche pas à calculer (contrairement à `int`). Ainsi, la différence entre

`evalf(Int(f,x=a..b))` et `evalf(int(f,x=a..b))`

est que dans le premier cas, la routine d'évaluation numérique d'intégrale est immédiatement appelée, alors que dans le second cas, le système cherche d'abord une forme symbolique. S'il en trouve une, par exemple lorsque $f = \cos x$, elle est employée ; sinon la routine d'évaluation numérique est appelée. Par exemple lorsque $f = \cos(\sin x)$, après avoir perdu du temps à chercher en vain une solution symbolique, le système réalise l'évaluation numérique. La fonction `Sum` joue le même rôle pour les sommes.

Lorsqu'une expression contient des formes inertes, la procédure `value` les rend actives afin de calculer la valeur de l'expression. Pour `RootOf`, la commande `allvalues` joue partiellement ce rôle.

Les principales formes inertes peuvent être récrites par `combine`, `expand`, `simplify` et certaines commandes du *package* `student`. Le tableau 7 montre les autres fonctions qui prennent en compte les formes inertes.

La forme inerte la mieux connue du système est `RootOf`. Son argument n'est pas forcément un polynôme ; les commandes `evalf` et `series` savent aussi traiter des équations plus générales. Voici par exemple une façon de trouver numériquement une racine de l'équation $z + \cos z = 2$

```
evalf(RootOf(z+cos(z)=2));
```

2.988268926

et voici comment obtenir un développement limité à l'origine de la fonction $y(x)$ définie implicitement par $y\exp(x) = y^5 + \ln(1+x)$ :

```
series(RootOf(y*exp(x)=y^5+ln(1+x),y),x);
```

$$x - \frac{3}{2}x^2 + \frac{4}{3}x^3 - x^4 + \frac{209}{120}x^5 + O(x^6)$$

**2.4. Hypothèses sur les variables.** Les variables non affectées posent problème lors des calculs. Nous avons rencontré un cas typique dans le §1 : que vaut $\sqrt{x^2}$ ? Si $x$ est réel positif, on voudrait obtenir $x$, si $x$ est réel négatif, on voudrait obtenir $-x$ et si $x$ est un nombre complexe quelconque, on doit choisir parmi deux racines complexes opposées. Ce problème du type de la variable se pose dans bien d'autres cas.

Dans le §1, le problème avait été réglé par l'emploi de l'option `symbolic` de la fonction `simplify` qui permet de réaliser les simplifications sans se poser de question. Dans le cas général, la bonne solution consiste à utiliser la fonction `assume` qui permet de préciser les propriétés d'une variable.

EXEMPLE 12. On aurait pu exprimer que les variables $R_1$ et $R_2$ étaient réelles positives de la façon suivante :

```
assume(R1>0); assume(R2>0);
about(R1);
    Originally R1, renamed R1~:
      is assumed to be: RealRange(Open(0),infinity)
```

La fonction `about` indique les hypothèses faites sur une variable. Après l'appel de `assume`, les variables sont automatiquement renommées et un tilde apparaît lors de leur impression. Mais on continue à les utiliser normalement en entrée.

EXEMPLE 13. Avec ces hypothèses sur $R_1$ et $R_2$ les simplifications auraient été réalisées automatiquement :

```
A:=normal(R1^2*(arccos(c1)-s1*c1)+R2^2*(arccos(c2)-s2*c2));
```

$$A := \arccos\left(\frac{1}{2}\frac{2R1^{\sim 2} - R2^{\sim 2}}{R1^{\sim 2}}\right) R1^{\sim 2} - \frac{1}{2}R2^{\sim}\sqrt{4R1^{\sim 2} - R2^{\sim 2}} + R2^{\sim 2}\arccos\left(\frac{1}{2}\frac{R2^{\sim}}{R1^{\sim}}\right)$$

Le seul moyen de supprimer les hypothèses faites sur une variable est de lui donner une valeur (qui peut être son nom). Par ailleurs les nouvelles hypothèses n'ont pas d'effet rétroactif et il faut éventuellement refaire des calculs.

EXEMPLE 14. La variable `e` garde sa valeur obtenue pour $a > 0$ même après avoir spécifié que $a$ est négatif.

```
assume(a>0); e:=1+sqrt(a^2);
```

$$e := 1 + a^{\sim}$$

```
assume(a<0); e;
```

TABLE 8 Principales propriétés reconnues par Maple.

| Déclaration | Fonctions l'exploitant |
|---|---|
| `assume(x<>0)` | `signum` |
| `assume(x>=0)` | `signum`, `abs`, `csgn`, `int` |
| `assume(x<=0)` | `signum`, `abs`, `csgn`, `int` |
| `assume(x>0)` | `signum`, `abs`, `csgn`, `int` |
| `assume(x<0)` | `signum`, `abs`, `csgn`, `int` |
| `assume(x,integer)` | `floor`, `frac`, `round`, `trunc` |
| `assume(x,real)` | `signum`, `abs`, `Re`, `Im` |

$$1 + a^{\sim}$$

```
e:=1+sqrt(a^2);
```

$$e := 1 - a^{\sim}$$

Pour résumer, voici les principales fonctions manipulant des hypothèses :

- `assume` déclare une hypothèse ;
- `additionally` rajoute une hypothèse à une variable ;
- `isgiven` teste si une hypothèse a été déclarée ;
- `is` teste si une hypothèse se déduit de celles qui ont été déclarées ;
- `about` liste les hypothèses faites sur une variable.

L'intégration de ce mécanisme n'est pas encore totale, mais Maple est le système le plus avancé dans ce domaine. Dans le tableau 8, nous indiquons les principales propriétés utilisables et une partie des fonctions qui en tiennent compte. Ce tableau est très incomplet, puisque les fonctions qui utilisent `signum` (fonction signe pour les expressions réelles ou complexes) par exemple tiennent également compte de ces propriétés. Il faut enfin noter que pour des raisons discutables d'efficacité, les tests internes sont faits par `isgiven` et non par `is`. Cela force parfois l'utilisateur à fournir des information redondantes.

EXEMPLE 15. Le polynôme $z^2 - 1$ est négatif dans l'intervalle $]-1, 1[$, mais la déduction n'est pas immédiate :

```
f:=signum(z^2-1):
assume(z<1,z>-1): f;
```

$$\mathrm{signum}(z^{\sim 2} - 1)$$

```
is(z^2<1);
```

$$true$$

```
additionally(z^2<1): f;
```

$$-1$$

**2.5. Objets composés.** Nous avons déjà utilisé des ensembles et des suites d'expressions dans le §1. Maple gère aussi des listes. Le tableau 9 décrit les caractéristiques et différences de ces objets. Un dernier type d'objet composé est la table, mais l'utilisation un peu délicate des tables en Maple repousse leur étude au chapitre II.

TABLE 9 Listes, ensembles et suites d'expressions.

| Objet | Caractéristiques |
|---|---|
| liste | type : `list`<br>syntaxe : `[a,b,c]`<br>ordonné : `[a,b,c]` $\neq$ `[b,a,c]`<br>éléments répétés : `[a,a,c]` $\mapsto$ `[a,a,c]`<br>plusieurs niveaux : `l:=[b,c];  [a,l,d]` $\mapsto$ `[a,[b,c],d]`<br>liste vide : `[]` |
| ensemble | type : `set`<br>syntaxe : {`a,b,c`}<br>non ordonné : {`a,b,c`} = {`b,a,c`}<br>pas d'éléments répétés : {`a,a,c`}$\mapsto${`a,c`}<br>plusieurs niveaux : `l:=`{`b,c`}; {`a,l,d`} $\mapsto$ {`a,`{`b,c`}`,d`}<br>ensemble vide : {} |
| suite d'expressions | type : `exprseq`<br>syntaxe : `a,b,c`<br>ordonné : `a,b,c` $\neq$ `b,a,c`<br>éléments répétés : `a,a,c` $\mapsto$ `a,a,c`<br>un seul niveau : `l:=b,c;  a,l,d` $\mapsto$ `a,b,c,d`<br>suite vide : `NULL` |

**2.6. Opérateurs fonctionnels.** Nous avons vu dans le §1 comment donner une valeur à une expression avec la fonction `subs`. Une autre façon de réaliser cette opération est d'utiliser un *opérateur fonctionnel*.

En Maple la notation flèche `->` crée une fonction de zéro ou plusieurs arguments.

EXEMPLE 16. Voici comment définir une fonction $f$ qui à $(x, y)$ associe l'expression $x^2 + y^2 - 1$

```
f:=(x,y)->x^2+y^2-1;
```

$$f := (x, y) \to x^2 + y^2 - 1$$

```
f(1,a);
```

$$a^2$$

La fonction $f$ s'évalue alors comme n'importe quelle fonction Maple.

Inversement, à partir d'une expression algébrique, on peut obtenir une fonction. Pour cela on utilise la fonction `unapply`, équivalent de l'abstraction du $\lambda$-calcul.

EXEMPLE 17. On définit la fonction `fA` ayant pour argument `R2` et calculant l'aire `A` obtenue dans le §1. Il est alors facile de vérifier les valeurs de l'aire pour $R_2 = 0$ et $R_2 = 2R_1$.

```
fA:=unapply(A,R2);
```

$$fA := R2 \to \pi {R1}^2 - \arccos\left(\frac{1}{2}\frac{-2{R1}^2 + {R2}^2}{{R1}^2}\right) {R1}^2 - \frac{1}{2} R2 \sqrt{4{R1}^2 - {R2}^2} + {R2}^2 \arccos\left(\frac{1}{2}\frac{R2}{R1}\right)$$

```
fA(0),fA(2*R1);
```

$$0, \pi {R1}^2$$

En Maple on peut même composer ou dériver les opérateurs fonctionnels à l'aide de l'opérateur de composition `@` et de l'opérateur de dérivation `D`. L'opérateur `@@` est l'opération de composition itérée. Ainsi, `f@f@f` et `f@@3` représentent tous deux la troisième itérée de $f$.

Exemple 18.

```
f:=x->x^x;
```

$$f := x \to x^x$$

```
(f@f)(x);
```

$$\left(x^x\right)^{\left(x^x\right)}$$

```
(D@@2)(f@f)(x);
```

$$\left((x^x)^{(x^x)}(\ln(x^x)+1)^2 + \frac{(x^x)^{(x^x)}}{x^x}\right)(x^x)^2(\ln(x)+1)^2 + (x^x)^{(x^x)}(\ln(x^x)+1)\left(x^x(\ln(x)+1)^2 + \frac{x^x}{x}\right)$$

Malgré toutes ces possibilités offertes par les opérateurs fonctionnels, nous conseillons de travailler plutôt sur des expressions algébriques pour manipuler des fonctions!— avec `subs` mathématiques (tab. 6 p. 25). Bien que l'évaluation soit un peu moins facile (il faut exécuter `subs` puis parfois `eval`), les expressions se prêtent à des opérations plus nombreuses. Par exemple, Maple sait calculer une primitive d'une expression, mais ne peut le faire pour un opérateur fonctionnel. Il en va de même pour de nombreuses commandes dont l'argument doit être une expression.

**2.7. Exercices.** Ces exercices visent essentiellement à faire exécuter des calculs simples en apprenant progressivement à se débrouiller à l'aide de la documentation en ligne. Pour chaque exercice, sont indiqués entre crochets les items de la documentation en ligne à consulter.

1. Calculer la valeur numérique de $e^{\pi\sqrt{163}} - 262537412640768744$ avec successivement 10, 20, 30, 40 et 50 chiffres significatifs. Commentaires. [`evalf`]
2. Montrer que $(z+1)(z+j)(z+j^2) = (1+z)(1+jz)(1+j^2z)$ où $j$ est une racine cubique de l'unité. Pour cela on définira $j$ comme un nombre algébrique. [`RootOf`, `evala`, `alias`]
3. Retrouver les formules développées de $\sin(3x)$, $\cos(3x)$ et $\mathrm{ch}(5x)$. [`expand`, `inifcns`]
4. Simplifier $\cos(4\arctan(x)) + \cos(6\arctan(x))$. [`expand`, `normal`]

5. Calculer le déterminant de
$$\begin{bmatrix} \cos(x+a) & \sin(x+a) & 1 \\ \cos(x+b) & \sin(x+b) & 1 \\ \cos(x+c) & \sin(x+c) & 1 \end{bmatrix}$$
et le simplifier. [`linalg`]
6. Utiliser le développement de TAYLOR de $\arctan x$ en $x = 0$ pour obtenir une approximation rationnelle[†] de $16\arctan(1/5) - 4\arctan(1/239)$. Faire l'évaluation numérique pour des développements d'ordre de plus en plus élevé. [`series`, `convert`, `subs`, `evalf`]
7. Résoudre $x^{\sqrt{x}} = (\sqrt{x})^x$. Tracer la courbe $y = x^{\sqrt{x}} - (\sqrt{x})^x$ pour $x \in [0, 5]$.
8. Résoudre le système d'équations suivant en $x$ et $y$ :
$$\begin{cases} xy & = a^2 \\ \ln^2 x + \ln^2 y & = \frac{5}{2}\ln^2 a \end{cases}$$
Résoudre le même système en remplaçant $a$ par 2. Comparer la forme des solutions. [`solve`]
9. Mener l'étude de la fonction $x \mapsto \sqrt{|x-1|} - \ln|x|$. Déterminer domaine de définition, extremums, asymptotes, points de rebroussement, concavité. [`singular`, `solve`, `diff`, `limit`, `plot`]
10. Obtenir l'expression
$$1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1+x}}}}}$$
à l'aide de la composition des fonctions.

[†]Cette formule, trouvée par MACHIN en 1706, lui permit de calculer 100 décimales de $\pi$.

CHAPTER II

# Utilisation avancée

Il n'est pas possible d'utiliser pleinement un système de calcul formel sans en comprendre les principes de fonctionnement. Une approche du type "boîte noire" où l'on interroge le système est très rapidement insatisfaisante. Il est nécessaire d'aider le système en cas d'échec et donc de comprendre ce pourquoi il a échoué.

La compréhension d'un système passe d'une part par la connaissance au moins vague des *algorithmes* employés, que nous détaillerons dans la seconde partie de ce livre. D'autre part, cette connaissance passe plus fondamentalement par l'assimilation du mode de fonctionnement des systèmes. Plus précisément, il est important de comprendre comment se fait le lien entre la connaissance mathématique et sa représentation informatique, afin de mieux appréhender ce qui est faisable et ce qu'il est illusoire d'attendre d'un système. Nous aborderons ces problèmes de représentation de l'information mathématique au §2, et nous verrons au §3 (notamment au §3.3) comment étendre le système. Auparavant, nous décrivons les notions de base du langage de programmation de Maple.

## 1. Primitives du langage de programmation

Le système Maple, comme la plupart des systèmes de calcul formel, est doté d'un langage de programmation. Bien souvent, un calcul nécessite l'utilisation de plusieurs instructions, qui dépendent éventuellement de résultats intermédiaires. Si ces opérations sont à effectuer sur de nombreuses données, il est alors utile d'en faire une ou plusieurs procédures, qui automatisent l'ensemble du calcul.

**1.1. Éléments de syntaxe.** Après une présentation rapide de la syntaxe du test, de l'itération et des procédures, nous reviendrons plus en détail sur ces deux dernières opérations et leurs nombreuses variantes.

*Itération.* L'itération est l'opération qui s'obtient dans la plupart des langages de programmation par les mots-clés *for* ou *while*. Il s'agit d'effectuer une même opération en faisant varier un indice, éventuellement avec une condition à vérifier avant de lancer une nouvelle étape.

La façon la plus directe d'effectuer une itération en Maple s'écrit :

**for** var $\left\{ \begin{array}{l} \textbf{from} \text{ départ } \textbf{by} \text{ pas } \textbf{to} \text{ fin} \\ \textbf{in} \text{ expr} \end{array} \right.$ **while** cond **do** calcul **od**

EXEMPLE 1. Voici la boucle qui calcule le pgcd de polynômes $a(x)$ et $b(x)$ par l'algorithme d'EUCLIDE :

```
p2:=a; t:= b;
while t<>0 do
   p1:=p2;
   p2:=t;
   t:=rem(p1,p2,x);
od;
pgcd:=p2;
```

Répétitivement, une division euclidienne est effectuée entre le quotient et le reste des deux polynômes précédents. Le dernier résultat non nul est le pgcd.

La syntaxe de la boucle sera illustrée sur d'autres exemples dans ce chapitre.

*Tests.* La syntaxe des tests est donnée par :

**if** cond1 **then** calcul1 **elif** cond2 **then** calcul2 **else** calcul3 **fi**

Cela signifie : "si la condition cond1 est vérifiée alors effectuer le calcul calcul1, sinon si la condition cond2 est vérifiée alors effectuer le calcul calcul2, et sinon le calcul calcul3". Le champ **elif** et le champ **else** sont bien entendu facultatifs, et le nombre de champs **elif** n'est pas limité.

EXEMPLE 2. En combinant l'itération et le test, voici comment calculer la somme des inverses des nombres premiers inférieurs à 10 :

```
s:=0:
for i to 10 do if isprime(i) then s:=s+1/i fi od:
s;
```

$$\frac{247}{210}$$

*Procédures.* Les opérateurs fonctionnels présentés au chapitre I constituent le cas le plus simple de procédures. La syntaxe générale de celles-ci suit le schéma suivant :

```
proc(x1:type1,x2:type2,x3:type3,...)
local y1,y2,y3,...;
global z1,z2,z3,...;
options op1,op2,op3,...;
   corps de la procédure
end;
```

Le corps de la procédure est constitué d'instructions du même type que les instructions de l'utilisation interactive. Nous reviendrons en détail sur les différentes parties de la procédure au §1.3. L'exemple suivant montre qu'il n'est pas difficile de créer une procédure à partir d'un calcul qui fonctionne.

EXEMPLE 3. Le calcul de pgcd de l'exemple 1 se transforme en une procédure qui calcule le pgcd de deux polynômes :

```
simple_gcd:=proc(a,b,x)
local p1,p2,t;
   p2:=a; t:= b;
   while t<>0 do
         p1:=p2;
         p2:=t;
         t:=rem(p1,p2,x);
   od;
   p2;
end:
```

**1.2. Itération.** L'itération en Maple existe sous quatre formes : la boucle `do`, introduite ci-dessus, et les primitives `seq`, `$`, `map` du langage. Nous passons maintenant en revue ces différentes instructions.

*Boucle do.* Seuls les mots-clés `do` et `od` sont obligatoires dans la boucle `do`. Nous allons illustrer la syntaxe du `do` sur plusieurs exemples, en commençant par la première version, plus proche des autres langages de programmation.

EXEMPLE 4. Le produit

$$\prod_{k=1}^{1000} (2k+1),$$

est calculé par la boucle

```
p:=1:
for t from 3 by 2 to 2001 do p:=p*t od:
```

Lorsque les champs `from` ou `by` ne sont pas précisés ils sont pris égaux à un, ce qui permet une expression assez condensée.

EXEMPLE 5. Le calcul des cinq premières dérivées de $\exp(e^x)$ est effectué par

```
f[0]:=exp(exp(x)):
for i to 5 do f[i]:=diff(f[i-1],x) od:
```

et si les dérivées d'ordre inférieur à cinq sont inutiles, il suffit d'exécuter

```
f:=exp(exp(x)):
to 5 do f:=diff(f,x) od:
```

Le champ `while` sert à contrôler l'arrêt, par exemple pour une itération numérique. Voici un exemple typique.

EXEMPLE 6. Le calcul de $\sqrt{2}$ est basé sur l'itération de NEWTON qui donne $\sqrt{a}$ comme limite de

$$u_{n+1} = \frac{1}{2}\left(u_n + \frac{a}{u_n}\right).$$

```
sqrt2:=2.:oldsqrt2:=0:
to 100 while abs(oldsqrt2-sqrt2)>10^(-8) do
   oldsqrt2:=sqrt2;
   sqrt2:=sqrt2/2+1/sqrt2 od:
sqrt2;
```

$$1.414213562$$

Le champ `to 100` est facultatif ici. Il est utilisé pour limiter le nombre d'itérations au cas où la suite ne convergerait pas.

Il est parfois utile de savoir qu'à la fin de l'exécution d'une boucle, la variable indice vaut la première valeur qui n'est pas utilisée dans le corps de la boucle. Si l'indice est utilisé ensuite, il faut penser à le réinitialiser. Mais cette valeur peut également être utile dans les calculs.

EXEMPLE 7. Dans l'exemple 4 ci-dessus, à la fin de la boucle, `t` vaut 2003.

EXEMPLE 8. Voici comment calculer la valeur minimale de $n$ telle que

$$\sum_{k=1}^{n} \frac{1}{k!} > 1,7$$

```
s:=0:
for k while s<=1.7 do s:=s+1/k! od:
nmin:=k-1;
```

$$nmin := 4$$

EXEMPLE 9. La commande `member` détermine si un objet appartient à une liste et, si c'est le cas, affecte à son troisième argument l'indice de la première occurrence. Voici comment elle pourrait s'écrire en Maple:

```
simple_member:=proc(a,l:list,pos:name)
local i, nb_elts;
   nb_elts:=nops(l);
   for i to nb_elts while a<>l[i] do od;
   if i=nb_elts+1 then false
   else pos:=i−1; true
   fi
end:
```

Il nous reste à décrire la seconde façon d'utiliser la boucle `do` : l'appel avec `in`. Dans ce cas, la variable d'indice, au lieu de prendre des valeurs entières dans une progression arithmétique, prend pour valeurs successives les *sous-expressions* de l'expression donnée. La définition précise de ce que l'on entend par sous-expression sera donnée en §2, disons simplement pour l'instant qu'il s'agit des opérandes de l'expression.

Exemple 10. Voici une manière de calculer la somme des carrés des éléments d'une liste :

```
s:=0:
for i in liste do s:=s+i^2 od:
```

En fait, `liste` dans cet exemple n'a pas besoin d'être une liste. Cette boucle donne le même résultat que `liste` vaille `[x,y,z]`, `x+y+z`, `x*y*z` ou `x,y,z`.

Signalons une erreur courante qui est commise lorsque l'objet `liste` est produit automatiquement (par exemple une suite d'expressions issue de `solve`). En effet, qu'on s'attende à une somme, un produit ou une suite d'expressions, lorsque `liste` contient une seule composante `x`, l'instruction ci-dessus ne calcule pas `x^2` comme on le voudrait, mais *la somme des carrés des composantes de* `x`.

Dans le cas de la suite d'expressions, ce problème se contourne aisément en mettant des crochets autour de celle-ci pour la transformer en liste. La différence entre

**for** i **in** s **do** ... **od** et **for** i **in** [s] **do** ... **od**

est que seule la seconde forme garantit un seul passage dans la boucle lorsque $s$ est une suite d'expressions à un élément. Il n'y a pas de solution similaire dans le cas de la somme et du produit ; il faut alors tester le type de l'objet avant la boucle.

L'exemple suivant illustre à quel point tout est optionnel dans la boucle en Maple.

Exemple 11. La boucle suivante calcule 2 indéfiniment.

```
do 1+1 od;
```

Ce type de boucle infinie est essentiellement utile en conjonction avec un test et l'instruction `break`. Cette instruction permet de sortir de la boucle sans attendre que l'indice ait atteint sa valeur maximale ou que la condition du champ `while` soit vérifiée. Ainsi l'analogue d'une boucle `repeat...until` de Pascal s'écrit en Maple

**do**
  ...
  **if** condition **then break fi**;
**od**;

*Procédure seq.* La boucle `do` est la boucle la plus générale de Maple. Dans quelques cas, elle est avantageusement remplacée par des commandes moins générales mais souvent plus efficaces. La plus utile de ces commandes est la procédure `seq`. Comme la boucle `do`, elle a deux syntaxes :

**seq**(f,i=a..b) ou **seq**(f,i=expr).

En termes de fonctionnalité, cette commande est équivalente selon le cas à l'une des deux boucles

$$\text{s:=NULL: } \textbf{for } \text{i} \begin{cases} \textbf{from } \text{a } \textbf{to } \text{b} \\ \textbf{in } \text{expr} \end{cases} \textbf{ do } \text{s:=s,f } \textbf{od}$$

Cependant le système connaît *a priori* la taille du résultat. Ceci permet de le construire sans utiliser de place mémoire supplémentaire pour les calculs intermédiaires et mène à un gain appréciable d'efficacité. Le coût de la construction passe ainsi de $O(n^2)$ à $O(n)$ où $n$ est le nombre d'éléments de la suite.

L'utilisation principale de la commande `seq` est le calcul de sommes ou de produits. Il suffit pour calculer

$$\sum_{i=a}^{b} f \qquad \text{ou} \qquad \prod_{i=a}^{b} f$$

d'exécuter

convert([**seq**(f,i=a..b)],‘+‘) ou convert([**seq**(f,i=a..b)],‘*‘).

L'avantage de cette forme d'itération par rapport à la boucle `do` est double. D'une part, comme pour la séquence, dans le cas d'une somme d'objets symboliques cette forme n'utilise pas de mémoire supplémentaire. D'autre part, les simplifications automatiques liées à la création de la somme ou du produit ne sont effectuées qu'une fois, au lieu d'être effectuées à chaque passage dans la boucle. Le coût de ces simplifications peut ainsi passer de $O(n^2)$ à $O(n \log n)$, où $n$ est le nombre d'opérandes du résultat.

En revanche, dans le cas où les objets sont numériques, la commande `seq` construit une liste parfois beaucoup plus grosse que la taille du résultat. Il faut donc choisir le type d'itération en fonction du type de $f$.

Exemple 12. L'exemple 4 est plus rapide que l'itération par `seq` :

```
convert([seq(2*k+1,k=1..1000)],‘*‘):
```

En revanche, le calcul suivant

```
s:=0: for n to 400 do s:=s+exp(n*x) od:
```

est quatre fois plus lent que sa version utilisant `seq` :

```
s:=convert([seq(exp(x*n),n=1..400)],‘+‘):
```

*Commande $.* La commande `$` produit efficacement deux types de séquences particulières : la répétition d'une expression et la succession d'entiers.

Dans le premier cas, la syntaxe `x$i` où `i` est un entier produit `i` copies de `x`. Dans le second cas, la syntaxe est simplement `$a..b`, où `a` et `b` sont des entiers, ce qui produit la suite d'expressions composée des entiers de `a` à `b`.

Exemple 13. Un emploi fréquent de la commande `$` est le calcul de dérivées. La procédure `diff` prend en effet comme premier argument la fonction et les arguments suivants sont les variables par rapport auxquelles il faut

calculer la dérivée. Pour calculer une dérivée $n^{\text{e}}$, la commande est abrégée par l'utilisation de `$`

```
f:=exp(x^2):
diff(f,x$5);
```

$$120xe^{x^2} + 160x^3e^{x^2} + 32x^5e^{x^2}$$

Ce qui n'est qu'une abréviation en utilisation interactive devient crucial dans l'écriture d'opérations plus complexes. L'instruction `diff(f,x$k)` simplifie l'écriture de nombreuses boucles. Par sa construction même, cette technique ne permet cependant pas de dériver zéro fois par rapport à une variable pour retrouver l'opérateur identité.

*Commande map.* Un peu comme `seq`, `map` applique une procédure à chacun des opérandes d'une expression. La différence tient en ce que `map` conserve le type de son argument. Si `s` est une somme, l'instruction `map(f,s)` est équivalente à

convert([**seq**(f(i),i=s)],§‘+‘);

L'application la plus fréquente est `map(f,l)`, où `l` est une liste, qui renvoie une liste dont chaque élément est le résultat de l'application de `f` à l'élément correspondant de `l`. Ceci permet d'appliquer une procédure à chacun des éléments d'une liste sans écrire de boucle. Si la procédure `f` prend plusieurs arguments, les autres arguments sont donnés après la liste.

Exemple 14. Voici un calcul simultané de dérivées :

```
l:=[sin(x),cos(x+y),exp(x^2)]:
map(diff,l,x);
```

$$[\cos(x), -\sin(x+y), 2xe^{x^2}]$$

La procédure `zip` est similaire mais beaucoup moins efficace car elle ne fait pas partie du noyau (voir tab. 4 p. 66). Elle permet de balayer simultanément deux listes.

**1.3. Procédures.** Comme l'ont fait apparaître les exemples précédents, la valeur retournée par une procédure est la dernière expression évaluée. Ceci permet d'encapsuler une suite d'opérations — mises au point en utilisation interactive — dans un `proc...end` sans avoir à rajouter d'indication particulière.

Bien que la syntaxe présentée p. 34 indique qu'il est possible de donner un type aux arguments d'une procédure, cela n'est nullement nécessaire. Lorsqu'un type est déclaré, à chaque appel de la procédure, l'interprète vérifie que l'argument est bien du type spécifié. Par exemple, à chaque exécution de la procédure `simple_member`, le deuxième argument et le troisième sont testés pour vérifier qu'il s'agit bien d'une liste et d'un nom de variable.

L'instruction `RETURN` permet de renvoyer un résultat avant d'arriver à la dernière instruction d'une procédure. De même, l'instruction `ERROR(mesg)` déclenche une erreur, avec `mesg` pour message.

EXEMPLE 15. Cette procédure renvoie son argument s'il est négatif et calcule sa factorielle sinon.

```
proc(n:integer)
local i,f;
  if n<0 then RETURN(n) fi;
  f:=1;
  for i to n do f:=f*i od;
  f
end;
```

EXEMPLE 16. Une autre façon de programmer la factorielle est d'utiliser la récursivité :

```
fact:=proc(n)
   if n<0 then ERROR('invalid argument') fi;
   if n=0 then 1
   else n*fact(n−1) fi
end:
```

Lorsque la procédure est appelée avec un entier positif, elle se rappelle avec l'argument réduit de 1, jusqu'à ce que l'argument vaille 0, alors elle renvoie 1 et effectue chacune des multiplications.

Les opérateurs fonctionnels introduits au chapitre I peuvent être utilisés dans le cas restreint où la procédure n'a pas de variable locale et est réduite à une seule instruction.

*Variables locales et globales.* Pour rendre les programmes réutilisables, il faut éviter qu'ils changent la valeur de variables de l'environnement. C'est à cela que servent les variables *locales* dans les procédures. Ces variables ne sont visibles que de l'intérieur de la procédure. Dès que la procédure termine ou qu'elle appelle une autre procédure, elles deviennent invisibles.

EXEMPLE 17. Cet exemple illustre la différence entre variable globale et variable locale :

```
y:=2: f:=proc() global y; y:=3 end:
f(): y;
                                3
y:=2: f:=proc() local y; y:=3 end:
f(): y;
                                2
```

Dans le premier cas, la variable globale `y` a été modifiée par la procédure. Dans le second cas, c'est la variable locale `y` qui a été modifiée, la variable globale restant inchangée. Un troisième exemple illustre ce qu'il en est des procédures appelées

```
a:=proc() local b,y;
   b:=proc() global y; y:=3 end;
```

```
   b()
end:
y:=2: a(): y;
```

$$3$$

Dans cet exemple, il y a deux procédures imbriquées, la première ayant pour variable locale `y`. L'affectation de `y` dans la sous-procédure `b` concerne la variable globale `y` qui de ce fait est modifiée. Contrairement à des langages comme Pascal, il n'y a pas moyen d'accéder depuis `b` à la variable locale `y` de `a`. Les seules variables accessibles depuis une procédure sont, outre ses variables locales, les variables définies au niveau de l'interprète.

Il faut faire attention à une erreur grave : laisser une variable locale sortir d'une procédure. Celle-ci s'affichera alors de la même façon que la variable globale de même nom, et il devient impossible de les distinguer.

Exemple 18. Voici la version la plus dépouillée de ce problème :

```
proc() local x; x end()-x;
```

$$x - x$$

Dans cet exemple, la différence `x-x` n'a pas été simplifiée, puisque les deux symboles `x` représentent des variables différentes, mais qui sont devenues toutes deux accessibles au *toplevel*.

Dans chaque procédure on dispose, outre des variables locales déclarées explicitement, de la variable `nargs` qui donne le nombre de paramètres passés à la procédure, de la suite d'expressions `args` des arguments (`args[1]` est le premier, `args[nargs]` le dernier) et comme en utilisation interactive, des variables `%`, `%%` et `%%%` qui donnent les dernières valeurs calculées *à l'intérieur de la procédure*.

*Passage des paramètres.* Le passage des paramètres se fait toujours par *valeur*, c'est-à-dire que les arguments sont toujours évalués avant d'être passés à une procédure[†]. Pour faire un passage par variable ou par référence, il faut passer le nom de la variable. C'est ainsi qu'une procédure peut modifier une variable.

Exemple 19. Voici un exemple de ce que l'on ne peut pas faire :

```
y:=2: f:=proc(x) x:=3 end: f(y);
 Error, (in unknown) Illegal use of a formal parameter
```

En effet, il y a erreur puisque l'on essaye d'effectuer `2:=3`. En revanche, en passant le nom de la variable en argument, il n'y a pas de problème :

```
y:=2: f('y'): y;
```

$$3$$

De nombreuses procédures Maple prennent ainsi des noms de variable en argument, comme `iquo` qui calcule le quotient de deux entiers et affecte le reste à la variable passée en troisième argument.

---

[†]Les rares exceptions sont `assigned`, `eval`, `evalf`, `evalhf`, `evaln`, `seq`, `time` et `userinfo`.

*Table de remember.* À toute procédure Maple est associée une table dans laquelle on peut stocker des couples (entrée, sortie) au cours d'une session. Lorsque la procédure est appelée, l'interprète commence par regarder si l'argument (ou la suite des arguments) est une entrée de la table, auquel cas il renvoie le résultat correspondant. Attention, ceci signifie qu'il ne faut pas utiliser cette table si la procédure tient compte lors du calcul de la valeur de variables globales. Par ailleurs, si une erreur se produit durant l'exécution de la procédure, rien n'est stocké dans la table de *remember*.

La façon la plus simple (et la plus brutale) d'utiliser cette table est d'ajouter au début de la procédure les mots-clés `option remember`. Dans ce cas tous les couples (entrée, sortie) seront gardés dans la table. Voici un exemple où l'intérêt de l'option *remember* est manifeste.

EXEMPLE 20. Cette procédure n'utilise pas l'option :

```
fibo1:=proc(n)
   if n=0 or n=1 then 1 else fibo1(n−1)+fibo1(n−2) fi
end:
time(fibo1(20));
```

25.683

Cette seconde procédure effectue le même calcul, mais avec l'option *remember* :

```
fibo2:=proc(n) option remember;
   if n=0 or n=1 then 1 else fibo2(n−1)+fibo2(n−2) fi
end:
time(fibo2(20));
```

.034

Dans le premier cas, le nombre d'opérations arithmétiques effectuées par le programme est exponentiel en $n$, dans le second il est linéaire.

En général, il faut faire un choix entre la vitesse que peut apporter une option *remember* et la consommation mémoire entraînée par la table de *remember*. Aussi vaut-il mieux limiter l'usage de cette option à des procédures qui travaillent sur des objets pour lesquels le temps de calcul est grand par rapport à la taille du couple (entrée, sortie).

Une meilleure utilisation de la table de *remember* consiste à n'y conserver qu'une sélection de ces couples. Il suffit pour cela d'affecter le résultat de la procédure, après que la procédure a été définie.

EXEMPLE 21. Voici une illustration du mécanisme :

```
f:=proc() 3 end: f(2):=4: f(2);
```

4

Ici `f` est la procédure constante 3, mais il est possible d'affecter une valeur quelconque à l'une de ses images, et le dernier appel montre bien que la procédure `f` n'est alors pas exécutée.

EXEMPLE 22. Voici une programmation de la factorielle plus proche de la définition mathématique :

fact:=**proc**(n) n*fact(n−1) **end**:
fact(0):=1:

Attention l'affectation `fact(0):=1` doit être faite *après* la définition de `fact`, car la création d'une procédure réinitialise sa table de *remember*.

EXEMPLE 23. Il est également possible de faire cette affectation à l'intérieur de la procédure, ce qui permet à la fin d'un calcul de stocker la valeur après avoir testé si cela en vaut la peine. Voici un exemple simplifié :

g:=**proc**(x) print(x); **if** x=2 **then** g(x):=3 **else** x **fi end**:

```
g(2); g(2);
```

2
3
3

Le premier appel `g(2)` effectue le calcul (l'instruction `print(x)` est exécutée) et entre le couple $(2, 3)$ dans la table de *remember* de `g`. Lors du second appel, le calcul n'est plus effectué et la procédure renvoie tout de suite la valeur.

C'est ainsi qu'en Maple fonctionnent la plupart des procédures correspondant à des fonctions mathématiques, lorsque leur argument est une variable non affectée. Par exemple, le corps de la procédure `exp` ressemble à ceci :

exp := **proc**(x)
  **if** type(x,float) **then** ...
  **elif** ...
  ...
  **else** exp(x):=éxp(x)'
  **fi**
**end**:

Il est impossible ici de renvoyer `exp(x)` ce qui entraînerait une boucle sans fin. Il serait possible de retourner simplement `'exp(x)'`, mais alors chaque apparition de ce terme dans une expression redéclencherait l'exécution de tout le corps de la procédure `exp`, ce qu'il est préférable d'éviter.

Quelle que soit la façon de modifier la table de *remember* (par option *remember* ou par affectation), il est possible de la voir et de l'utiliser en faisant `op(4,eval(.))`.

EXEMPLE 24. Voici le contenu de la table de *remember* de la procédure `exp` :

```
op(4,eval(exp));
```

table([
$-\infty = 0$
$x = e^x$
$I\pi = -1$
$0 = 1$
])

Initialement, cette table contient des valeurs qui ont été rentrées explicitement par des affectations. Le couple `(x, exp(x))` a été introduit par l'invocation de la procédure `exp` sur la variable `x`.

Pour conclure, mentionnons un emploi important de l'option *remember* en Maple, à savoir la procédure `readlib`, qui ne charge ainsi qu'une seule fois les procédures en mémoire.

### 1.4. Exercices.

1. Calculer numériquement exp(1) sans utiliser `evalf`, par la formule
$$\exp(1) = \sum_{n=0}^{\infty} \frac{1}{n!},$$
en arrêtant le calcul lorsque le terme devient inférieur à $10^{-\texttt{Digits}-2}$.
2. La suite de Syracuse est la suite définie par $u_0 \in \mathbb{N}^\star$, et
$$u_{n+1} = \begin{cases} 1 & \text{si } u_n = 1, \\ u_n/2 & \text{si } u_n \text{ est pair}, \\ 3u_n + 1 & \text{sinon.} \end{cases}$$
On ne sait pas à l'heure actuelle s'il existe un entier $u_0$ pour lequel cette suite n'atteint jamais 1.
   (1) Écrire une procédure prenant en argument un entier positif et calculant le nombre minimum d'itérations nécessaires pour aboutir à 1 à partir de cet entier ;
   (2) écrire une procédure similaire agissant sur un ensemble d'entiers, et renvoyant le nombre minimum d'itérations nécessaires pour que tous les entiers de l'ensemble aboutissent à 1 (sans appeler la procédure précédente).
3. Écrire une procédure récursive prenant en entrée une liste et renvoyant la liste de toutes les permutations des éléments de cette liste.
4. Écrire une procédure prenant en entrée deux entiers positifs $k$ et $n$ et renvoyant
$$\sum_{1 \leq i_1 \leq i_2 \leq \cdots \leq i_k \leq n} \frac{1}{i_1 \cdots i_k}.$$
[Indication : gérer une liste des indices.]

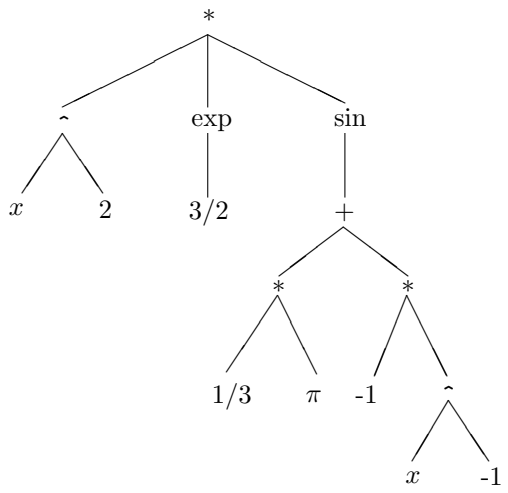



FIGURE 1 L'arbre représentant $x^2 \exp(3/2) \sin(\pi/3 - 1/x)$.

## 2. Manipulation d'expressions

Les procédures que nous avons vues jusqu'à maintenant ne font pas vraiment de calcul formel, faute d'outil pour manipuler des expressions. Dans cette section nous allons étudier en détail la notion d'expression, et ainsi arriver à une meilleure compréhension du fonctionnement intime de Maple.

La notion d'expression est au cœur du calcul formel. Lorsque l'on utilise une expression mathématique telle que $\sin(\pi/3 + x)$, on a en tête une fonction (sin) et un réel $\pi$ dont on connaît de nombreuses propriétés, ainsi qu'une variable $x$. Du point de vue du système, en revanche, il s'agit d'une *expression*. Pour le système, *a priori*, il s'agit d'une fonction nommée `sin` appliquée à une somme de deux termes, à savoir une variable $x$ et le quotient d'*une autre variable* $\pi$ par l'entier 3. Pour toutes les opérations mathématiques que l'utilisateur souhaite réaliser sur cette expression (la dériver, l'intégrer, calculer son développement de TAYLOR), le système ne dispose que de cette information syntaxique.

Des questions centrales du calcul formel, comme la simplification, sont liées à cette distinction entre la vision syntaxique du système et la vision mathématique de l'utilisateur. Ce livre s'attache à décrire les fonctionnalités des systèmes en termes de *classes d'expressions*. Ces classes sont des ensembles d'expressions purement syntaxiques et non mathématiques. Ainsi, 0 et $1 - \exp(1)\exp(-1)$ sont deux objets identiques d'un point de vue mathématique, mais très différents du point de vue d'un système de calcul formel.

Un des objectifs de ce chapitre est de montrer comment le système Maple réalise le passage du syntaxique au mathématique. Les autres systèmes (excepté Axiom) fonctionnent sur les mêmes principes.

**2.1. Structure des expressions.** Une expression comme

$$x^2 \exp(3/2) \sin(\pi/3 - 1/x)$$

peut être visualisée comme un arbre (fig. 1). Cet arbre se lit de haut en bas et se décrit avec un vocabulaire généalogique : les nœuds situés sous la racine sont ses fils, et les fils d'un même nœud sont des frères.

À la racine de l'arbre se trouve l'opérateur `*` qui représente le produit. Ses trois fils sont les trois facteurs $x^2$, $\exp(3/2)$, $\sin(\pi/3 - 1/x)$, eux-mêmes représentés par des arbres. Les nœuds de l'arbre, comme la racine, représentent soit des opérations (`+`,`*`, `^`), soit des fonctions (sin, exp), et les feuilles représentent soit des rationnels (2, 3/2, 1/3, −1), soit des noms de variables (`x`, `Pi`).

Les procédures Maple travaillent toutes sur des expressions représentées par de tels arbres et les opérations de base consistent donc à analyser ces arbres pour en déterminer les composants, à modifier ces composants et à les utiliser pour créer de nouveaux arbres.

*Opérandes.* Les arbres qui se trouvent sous la racine peuvent être isolés par la procédure `op` (comme opérande). Avec une expression comme argument, cette procédure renvoie la suite des fils de l'expression, leur nombre étant renvoyé par la procédure `nops`. Pour sélectionner un fils particulier, on donne en premier argument à `op` son numéro. On peut bien sûr à nouveau isoler un sous-arbre d'un sous-arbre. Avec 0 comme premier argument, `op` renvoie la racine de l'arbre (sauf pour les séries et les tables).

Exemple 25. L'expression ci-dessus servira d'exemple pour la majeure partie de cette section.

```
f:=x^2*exp(3/2)*sin(Pi/3-1/x):
op(f);
```

$$x^2, \quad e^{3/2}, \quad \sin(\pi/3 - 1/x)$$

```
op(1,op(op(3,f)));
```

$$\pi/3$$

```
op(0,op(3,f));
```

$$\sin$$

```
op(0,f);
```

$$*$$

L'ordre des opérandes dans une somme ou un produit n'est pas nécessairement le même d'une session à l'autre. Il ne faut donc pas baser un programme sur la position d'un nœud par rapport à ses frères dans un arbre dont la racine est un opérateur commutatif. La seule propriété garantie, c'est que le coefficient rationnel d'un produit lorsqu'il est différent de 1 est la première opérande.

*Sous-expressions.* Un sous-arbre est l'ensemble des descendants d'un nœud de l'arbre. On appelle *sous-expression* une partie d'expression qui forme un sous-arbre de l'arbre représentant l'expression. Dans notre exemple, $x$, 2, $3/2$, $\sin(\pi/3-1/x)$, $x^2$, $\pi/3-1/x$ sont des sous-expressions, mais pas $x^2\exp(3/2)$, qui n'est pas représenté par un sous-arbre (il manque un des descendants du produit).

La commande `has` détermine si une expression apparaît en sous-expression d'une autre. L'utilisation de `has` permet par exemple de déterminer si une expression est indépendante d'une variable.

EXEMPLE 26. Toujours sur la même expression :

```
has(f,x), has(f,z);
```

true, false

Contrairement à `op` et `nops`, `has` parcourt toute l'expression (en cas de réponse négative), ce qui est coûteux sur de très grosses expressions. Pour déterminer si une des opérandes est égal à une expression, il vaut mieux utiliser `member` :

```
member(x^2,[op(f)]);
```

true

L'instruction `[op(f)]` crée une liste dont les éléments sont les opérandes de `f`. Il devient possible d'y appliquer `member`, qui teste ici l'appartenance de $x^2$ à cette liste.

**2.2. Création d'expressions.** Les opérations que nous venons de voir permettent d'*explorer* une expression. Celles qui suivent créent des expressions à partir d'expressions existantes. Combinées avec les primitives de la première section, elles forment tout l'arsenal du programmeur Maple.

Les principales procédures qui créent de nouveaux objets à partir d'expressions existantes sont :

`subsop`, `convert`, `subs`, `select`.

Il faut noter que ces procédures *ne modifient pas* leur argument. La plupart des objets que manipule Maple ne peuvent en effet pas être modifiés. Les procédures que nous allons présenter créent des copies altérées de leur argument. Cette copie n'entraîne cependant pas une trop grande consommation de mémoire, pour des raisons sur lesquelles nous reviendrons au §3.2.

*Procédure subsop.* La procédure `subsop` remplace un opérande par une valeur.

EXEMPLE 27.

```
liste:=[2,4,5,8]:
subsop(3=6,liste);
```

$[2,4,6,8]$

Une utilisation importante de `subsop` est la suppression d'un élément en exploitant le fait que `NULL` est l'élément neutre des suites d'expressions.

Exemple 28. Toujours sur la même liste (on remarque au passage qu'elle n'a pas été modifiée par la commande `subsop`) :

```
subsop(1=NULL,liste);
```

$$[4, 5, 8]$$

*Procédure convert.* Nous avons déjà rencontré la procédure `convert` pour calculer des sommes et des produits, en utilisation conjuguée avec `seq`. De même que `subsop` permet de changer un fils de l'arbre représentant l'expression, `convert` permet d'en changer la racine. Son second argument est le nouveau type. La principale utilisation de `convert` est celle que nous avons vue avec `seq`, rendue nécessaire par l'inexistence d'une addition et d'un produit préfixes en Maple.

*Procédure subs.* La façon naturelle de calculer en Maple la valeur d'une fonction pour une valeur donnée de sa variable consiste à substituer partout dans l'expression la variable par sa valeur. Les règles de simplification automatique amènent alors à la valeur cherchée.

Exemple 29. Nous évaluons un polynôme :

```
f:=x^2+3*x+7;
subs(x=3/2,f);
```

$$\frac{55}{4}$$

Plus généralement, `subs` permet de remplacer toute sous-expression par une expression syntaxiquement admissible.

Exemple 30. Il ne faut pas oublier qu'une sous-expression est l'ensemble des descendants d'un nœud de l'arbre. Ainsi, lors de la substitution

```
subs(a+b=3,a+b+c+sin(a+b));
```

$$a + b + c + \sin(3)$$

le premier terme $a + b$ n'est pas remplacé, puisqu'il ne constitue pas une sous-expression de la somme. Ce type de simplification peut s'obtenir soit par

```
subs(a=3-b,a+b+c+sin(a+b));
```

$$3 + c + \sin(3)$$

soit par le mécanisme plus général

```
simplify(a+b+c+sin(a+b),{a+b=3});
```

Cette dernière commande est cependant extrêmement coûteuse de par l'algorithme employé (voir chap. VI).

Comme nous l'avons déjà mentionné au chapitre I, il est parfois nécessaire d'appliquer `eval` au résultat de `subs`. Les raisons de cette nécessité seront éclaircies au §3.1.

*Procédure select.* Une autre commande très utile pour isoler des sous-expressions particulières est la procédure `select`. Elle prend en argument une procédure qui renvoie un booléen (comme `type` ou `has`), puis une expression et éventuellement d'autres arguments de la procédure. Elle renvoie l'expression en conservant uniquement les opérandes pour lesquels la procédure donnée en premier argument retourne `true`.

EXEMPLE 31. Voici quelques applications à une liste :

```
liste:=[sin(x),3,Pi/2,26]:
select(has,liste,x), select(type,liste,integer);
```

$$[\sin(x)], \qquad [3, 26]$$

EXEMPLE 32. Pour demander en une ligne à Maple de calculer

$$\sum_{\substack{p \text{ premier} \\ p \leq 1000}} \frac{1}{2^p},$$

il suffit de faire

```
convert([seq(1/2^p,p=select(isprime,[$1..1000]))],`+`);
```

**2.3. Types de base et simplification automatique.** En Maple, la racine de l'arbre détermine le *type* de l'expression. Les types de base du système sont résumés au tableau 1. Par exemple, la liste `[1,2,3]` est représentée par un arbre dont la racine est le mot-clé `list` et les fils sont 1, 2, 3.

Pour certains de ces types, quelques simplifications automatiques ont lieu à la création de l'expression. Nous détaillons maintenant les principales simplifications.

*Sommes et produits.* Les sommes sont traitées comme des sommes de produits (nombre fois expression), où un nombre est de type `numeric`, c'est-à-dire `integer`, `rational` ou `float`. De même, les produits sont traités comme des produits de puissances (expression puissance nombre). À la création d'une somme ou d'un produit, les opérandes sont parcourus pour regrouper les sous-expressions égales[†]. Ainsi, pour évaluer la somme $2x + 3y + 3x$, le simplificateur interne commence par passer en revue les opérandes pour s'apercevoir que $x$ apparaît deux fois, et regroupe alors en effectuant l'opération sur les `numeric` 2 et 3. Le produit $x^2y^3x^3$ est traité exactement de la même manière. Sont traitées à part les expressions numériques dans les produits : elles sont toujours groupées en tête du produit.

EXEMPLE 33. Le produit $x^2 \times 3y^2 \times 4x^{3/2}yz$ est créé ainsi :

```
x^2*3*y^2*4*x^(3/2)*y*z;
```

[†]Cette opération n'est pas aussi coûteuse que l'on pourrait le penser car le test d'égalité des expressions se fait en temps constant indépendamment de la taille des expressions, sans parcourir les deux arbres récursivement (voir §3.2).

TABLE 1 Types de base du système.

| Nom du type | Usage |
|---|---|
| `+`, `*`, `^` | opérations |
| `integer` | entiers |
| `rational` | rationnels |
| `float` | nombres à virgule flottante |
| `string` ou `name` | chaînes de caractères ou noms de variable |
| `function` | fonctions |
| `list` | listes |
| `set` | ensembles |
| `exprseq` | suites d'expressions |
| `=` | équations |
| `<>` | inéquations |
| `<`, `<=` | inégalités strictes et larges |
| `and`, `or`, `not` | expressions booléennes |
| `..` | intervalles |
| `procedure` | procédures |
| `table` | tables |
| `series` | développements limités |
| `.` | concaténation de chaînes de caractères |
| `indexed` | variables indexées comme `a[3]` |
| `uneval` | expressions protégées (voir §3.1) |

$$12y^3x^{7/2}z$$

Cet exemple illustre également l'absence d'ordre automatique dans les produits et les sommes.

*Listes et ensembles.* La distinction entre listes et ensembles provient de ce que dans ces derniers les éléments n'apparaissent qu'une fois (tab. 9 p. 29). À la création d'un ensemble, le simplificateur interne entreprend un parcours similaire à celui qu'il effectue pour les sommes et les produits, afin d'éliminer les doubles. Une procédure effectuant l'union de deux ensembles s'écrit donc simplement :

```
Union:=proc(e1:set,e2:set) {op(e1),op(e2)} end:
```

C'est d'ailleurs ainsi que la procédure `union` était écrite dans les premières versions de Maple.

En conséquence, lorsque les objets manipulés sont tous distincts, il vaut mieux utiliser des listes plutôt que des ensembles, puisque la création d'une liste ne gaspille pas de temps dans cette simplification automatique.

*Suites d'expressions (le type* `exprseq`*).* La seule simplification se produisant à la création d'une suite d'expressions est la suppression de l'élément neutre

`NULL`, dont nous avons montré comment tirer parti par la commande `subsop` (p. 48).

*Développements limités (*`series`*).* Ce type est un peu particulier comme le montre l'action de `op` :

```
s:=series(sin(z),z);
```

$$s := z - \frac{z^3}{6} + \frac{z^5}{120} + O(z^6)$$

```
op(s);
```

$$1, 1, -1/6, 3, 1/120, 5, O(1), 6$$

Les opérandes sont successivement des coefficients et des puissances, le dernier terme utilisant l'identité $O(z^6) = O(1)z^6$. Le nom de la variable est accessible comme $0^{\text{e}}$ opérande. Lors de la création d'un développement les termes dont le coefficient est 0 sont supprimés. Cette simplification est utile pour rechercher un développement par coefficients indéterminés (voir chap. VIII).

### 2.4. Types plus complexes.

*Types du système.* Les 26 types de base du tableau 1 p. 50 sont insuffisants en pratique. Par exemple, un programme de calcul de factorielle a besoin de s'assurer que son argument est un entier positif ou nul. Pour cela il existe une procédure `type` qui reconnaît 84 types plus complexes, comme `posint` pour les entiers positifs, `nonnegint` pour les entiers positifs ou nuls, `numeric` pour les entiers, rationnels ou flottants,...

Tous ces types sont utilisables pour la spécification des arguments des procédures et sont aussi accessibles via la commande `type`.

*Types procéduraux.* Il est par ailleurs possible de *créer* des types. Un nouveau type est déclaré par la création d'une procédure `type/nom_de_type`. Cette procédure renvoie `true` ou `false` selon que son argument est du type `nom_de_type` ou non.

EXEMPLE 34. Le type "entier positif ou nul inférieur à 100" est créé par :

```
§'type/int100':=proc(f) evalb(type(f,nonnegint) and f<100) end:
```

Ce type s'utilise comme n'importe quel type Maple :

```
smallfact:=proc(n:int100)
local f,i;
   f:=1;
   for i to n do f:=f*i od
end:
smallfact(0):=1:
smallfact(4);
```

24

```
smallfact(100);
Error, smallfact expects its 1st argument, n, to be of type
int100, but received 100
```

Les types peuvent également être paramétrés. Ainsi le type `polynom` de Maple peut prendre en argument une variable et le type des coefficients. Pour tester un tel type, la procédure `type/nom_de_type` utilise des arguments supplémentaires.

EXEMPLE 35. Le type suivant permet de détecter si une expression (mais pas nécessairement la fonction qu'elle définit) est indépendante d'une variable donnée.

```
§'type/independent':=proc(f,x) not has(f,x) end:
type(x^2+3*y,independent(x));
```

false

L'intérêt d'un tel type réside dans son emploi avec `select` ou lors de la création de types structurés qui sont décrits au paragraphe suivant.

Cet exemple nous permet par ailleurs d'introduire la distinction entre types de surface et types récursifs. Pour savoir si une expression est de type `+`, il suffit de regarder sa racine. En revanche, pour savoir si une expression est du type `independent` ci-dessus, ou du type `constant`, il faut parfois parcourir toute l'expression. Si l'on recherche l'efficacité, il faut donc toujours se demander si les caractéristiques à reconnaître ne peuvent pas être obtenues en "restant à la surface" de l'expression, c'est-à-dire près de la racine.

*Types structurés.* Le langage des types structurés permet de construire des types complexes à partir de types existants, qu'ils soient de base ou procéduraux. Par exemple, une liste contenant exclusivement des entiers ou des variables sera sélectionnée par le type `list({integer,name})`. La syntaxe détaillée des types structurés est obtenue par `?type[structured]`. Ces types évitent souvent l'écriture d'une procédure, d'où des tests beaucoup plus efficaces.

EXEMPLE 36. Pour dériver une expression de type puissance par rapport à une variable $x$ en appliquant

$$(f^a)' = af'f^{a-1},$$

il faut d'abord vérifier que $a$ ne dépend pas de la variable $x$. L'écriture

**if** type(expr,§'^') and not has(op(2,expr),x) **then** ...

est avantageusement remplacée par

**if** type(expr,anything^independent(x)) **then** ...

qui utilise le type `independent` que nous venons de créer et le type `anything` satisfait par n'importe quel objet à l'exception des suites d'expressions.

*Type des sous-expressions.* La commande `hastype`, analogue à `has`, teste si des sous-expressions ont un type donné. La commande `indets` effectue le même parcours de l'expression, tout en collectant l'ensemble des sous-expressions du type demandé.

EXEMPLE 37. Pour déterminer si une expression contient des exponentielles qui peuvent se développer, il suffit d'utiliser un type structuré.

```
f:=1+2*x*sin(exp(x+y+exp(x+2)));
```

$$f := 1 + 2x\sin(e^{x+y+e^{x+2}})$$

```
hastype(f,exp('+'));
```

$$true$$

```
indets(f,exp('+'));
```

$$\left\{e^{x+y+e^{x+2}}, e^{x+2}\right\}$$

Il faut noter que `indets` parcourt toute l'expression récursivement et appelle `type` sur toutes les sous-expressions (idem pour `hastype` en cas d'échec). Si le type testé est lui-même un type récursif et non un type de surface, le test peut s'avérer peu performant.

**2.5. Exercices.**

1. Écrire les procédures suivantes qui s'appliquent à des listes :
   (1) `append` qui renvoie la concaténation de plusieurs listes ;
   (2) `cons` et `endcons` qui renvoient la liste après avoir ajouté un élément respectivement au début et à la fin ;
   (3) `first` et `last` qui renvoient respectivement le premier et le dernier élément d'une liste ;
   (4) `reverse` qui renvoie une liste dont les éléments sont dans l'ordre inverse de son argument ;
   (5) `rest` qui renvoie la fin d'une liste à partir du $i^{\text{e}}$ élément.
2. La structure réelle des expressions est légèrement plus compliquée que ce qui apparaît en figure 1 p. 45. On peut s'en rendre compte en faisant `subs(1=toto,f)`, où `f` est l'expression qui y est représentée. Expliquer ce qui apparaît. Refaire l'expérience avec une somme au lieu d'un produit.

## 3. Approfondissement du système

Dans cette section nous abordons des particularités de Maple qu'il est bon de comprendre pour l'utiliser intensivement. Nous conseillons de passer directement à la seconde partie du livre, pour revenir sur cette section après avoir acquis une bonne pratique du système.

**3.1. Règles d'évaluation.** La commande `subs` (§2.2) ne fait que substituer un terme à une sous-expression. Les simplifications automatiques sont effectuées, mais l'expression n'est pas réévaluée.

Exemple 38.

```
f:=sin(x)+2*x+3:
g:=subs(x=0,f);
```

$$g := \sin(0) + 3$$

La variable $x$ a bien été remplacée par 0, mais la procédure `sin` n'a pas été rappelée pour simplifier le résultat. Malgré ce résultat dans lequel apparaît $\sin(0)$, la variable `g` semble bien avoir été simplifiée :

```
g;
```

$$3$$

Pour comprendre ce qui se produit dans cet exemple, il est important de connaître certaines règles d'évaluation de Maple. Les règles d'évaluation d'un langage de programmation spécifient quand et comment les expressions sont évaluées. Nous allons maintenant préciser ces règles pour le système Maple.

3.1.1. *Évaluation et protection des variables*

La différence principale entre Maple et les langages de programmation traditionnels réside dans l'évaluation des variables non affectées. Dans les langages traditionnels, cette évaluation est considérée comme une erreur. En Maple, elle renvoie le nom de la variable. Ce mécanisme permet de traiter les indéterminées mathématiques comme des variables informatiques.

En pratique il est recommandé de bien distinguer ces deux types d'objets. Le membre gauche d'une affectation est réservé aux variables informatiques qui ont ainsi toujours une valeur. Le membre droit peut faire apparaître à la fois des variables informatiques, qui sont alors évaluées à leur valeur, et des indéterminées mathématiques. L'évaluation d'une expression pour certaines valeurs des indéterminées est obtenue grâce à `subs`, comme dans l'exemple ci-dessus.

L'affectation se visualise comme la création d'une flèche de la variable vers la valeur. Par exemple, la suite d'affectations

```
x:=y: y:=z:
```

crée les flèches représentées en figure 2. La première commande crée l'expression formée de l'indéterminée `y` et lui donne pour nom `x`. La seconde utilise l'indéterminée `y` comme une variable et lui affecte l'indéterminée `z`.

$$x \to y \to z$$

FIGURE 2 Évaluation de variables.

L'évaluation consiste alors à suivre ces flèches pour obtenir une valeur. Pour décomposer les étapes de l'évaluation, nous utilisons la commande `eval` dont le second argument contrôle le nombre de flèches suivies :

```
x,eval(x,1),eval(x,2);
```

$$z, y, z$$

Le premier résultat montre que l'évaluation normale suit toutes les flèches. Le deuxième montre qu'en réalité, la valeur de `x` est celle de la variable `y`. Si `y` est modifiée, sa nouvelle valeur sera aussi celle que renverra l'évaluation de `x`.

Contrairement à l'égalité mathématique, l'affectation est dissymétrique puisqu'elle crée une flèche de la variable vers la valeur. Une autre cause de cette dissymétrie tient à l'évaluation. Le membre droit de l'affectation est évalué, le membre gauche ne l'est pas.

EXEMPLE 39. Considérons les instructions

```
x:=y: y:=x:
```

La première affectation crée une flèche de `x` vers la valeur de `y`. Ensuite, le membre droit de la seconde affectation s'évalue en la valeur de `x`, c'est-à-dire celle de `y`. Cette seconde affectation n'a donc aucun effet puisqu'elle crée une flèche de `y` vers sa propre valeur.

Pour désaffecter une variable, il faut lui affecter son nom. Le rôle des apostrophes est de protéger leur argument contre une évaluation. L'affectation

```
x:='x':
```

est traitée normalement : le membre droit est d'abord évalué. Mais les apostrophes ont une règle d'évaluation particulière. Le résultat de l'évaluation d'une expression entre apostrophes est cette expression non évaluée. Dans la

$$'x'$$
$$\downarrow$$
$$x \to 3$$

FIGURE 3 Évaluation d'une variable protégée.

désaffectation, la valeur du membre droit est donc la variable elle-même (fig. 3). Chaque évaluation fait disparaître un niveau de protection :

```
x:=3: v1:=''x''; v2:=v1; v3:=v2;
```

$$v1 :='x'$$
$$v2 := x$$
$$v3 := 3$$

La protection permet également de créer des boucles (généralement involontaires).

EXEMPLE 40. Une variation de l'exemple 39 crée l'exemple le plus simple de boucle :

```
x:=y: y:='x':
eval(x,10), eval(x,17);
```

$$x, y$$

En revanche, l'évaluation de `x` ou `y` sans limitation sur le nombre d'étapes entraîne l'évaluateur dans une boucle infinie qui se termine par le débordement de la pile et sur certaines machines par la mort du système.

L'utilisation conjointe de la commande `eval` et de la protection permet un accès simultané à la variable et à sa valeur.

EXEMPLE 41. La procédure `incr` augmente de 1 la valeur d'une variable :

incr:=**proc**(x:name) **if** assigned(x) **then** x:=eval(x)+1 **fi end**:

Il est nécessaire de s'assurer que l'argument est bien un nom de variable affectée, car autrement `eval(x)` renverrait le nom $x$ et en termes de flèches, l'affectation construirait donc une flèche de $x$ vers un arbre dont la racine est `+`, dont un opérande est 1 et l'autre opérande est à nouveau $x$. À l'évaluation suivante de cet arbre, le système mourrait d'une explosion de la pile en parcourant indéfiniment ces flèches.

```
x:=3: incr(x);
Error, incr expects its 1st argument, x, to be of type name,
but received 3
incr('x'): x;
```

Pour désaffecter une variable, il faut lui affecter son nom. La protection permet d'avoir accès à ce nom mais est parfois délicate d'emploi. La commande `evaln`, plus générale, donne un accès direct à ce nom pour toutes les variables.

EXEMPLE 42. Pour désaffecter les variables indexées `x[1]`...`x[10]`, on ne peut pas faire

```
for i to 10 do x[i]:='x[i]' od;
```

puisque la variable `i` ne va pas être évaluée dans le terme de droite. En revanche, on peut utiliser `subs` ainsi

```
for i to 10 do x[i]:=subs(t=i,x[t]) od;
```

La solution la plus directe consiste à utiliser la commande `evaln` :

```
for i to 10 do x[i]:=evaln(x[i]) od;
```

### 3.1.2. *Évaluation des expressions*

L'évaluation d'une expression consiste à parcourir l'arbre la représentant en évaluant chacun des nœuds et chacune des feuilles. Les simplifications automatiques sont d'abord appliquées à l'expression. En dehors des cas spéciaux d'évaluation (comme les apostrophes), les descendants de la racine sont évalués récursivement et la procédure qui est à la racine est alors évaluée avec ces arguments.

EXEMPLE 43. Pour illustrer la simplification automatique de la première phase, nous créons une procédure dont l'évaluation produit une boucle infinie :

```
boucle_inf:=proc() do od end:
```

Voici quelques simplifications automatiques effectuées avant l'évaluation :

```
true or boucle_inf(), boucle_inf()-boucle_inf();
```

$$true, \quad 0$$

La protection des expressions agit comme pour les noms : l'évaluateur se contente d'ôter un niveau de protection, mais n'évalue pas l'objet protégé. Les simplifications automatiques ont cependant lieu :

```
f('boucle_inf()'),'sin(boucle_inf()-boucle_inf())';
```

$$f(\text{boucle\_inf}()), \quad \sin(0)$$

Dans le contexte du calcul symbolique, la protection est cruciale. Par exemple, il est absolument nécessaire d'avoir un mécanisme qui permette à la procédure `exp` appelée avec le nom `x` de renvoyer `exp(x)` sans que celui-ci soit à nouveau évalué. La valeur retournée est donc `'exp(x)'`. Nous avons vu au §1.3 que cette valeur était même mise dans la table de *remember* de la procédure `exp`, mais ceci ne constitue qu'une optimisation et n'est pas absolument nécessaire.

Pour conclure, nous mentionnons un usage constant fait par Maple de la protection, à savoir le chargement automatique de procédures. Au début d'une session Maple, un certain nombre de procédures sont définies par

```
fct:='readlib('fct')':
```

Ainsi lorsque `fct` est appelée pour la première fois, la procédure est chargée en mémoire. Les apostrophes externes sont là pour retarder l'appel à `readlib`, les apostrophes internes servent à ce que lors de l'appel, l'argument de `readlib` soit le nom de la procédure et non la valeur `readlib(fct)`, ce qui entraînerait une boucle infinie.

EXEMPLE 44. La procédure `factor` est définie ainsi :

```
eval(factor,1);
```

readlib('factor')

```
print(factor);
   proc(x,K) ... end
```

Dans le second appel, l'argument de `print` est d'abord évalué et la définition initiale de la procédure a été remplacée par le contenu trouvé dans la bibliothèque.

### 3.1.3. *Évaluation totale ou partielle*

Une question qui se pose à Maple est de savoir à quel moment évaluer les expressions. Si l'évaluation d'une expression est déclenchée chaque fois qu'elle est rencontrée, le système risque de parcourir tout l'arbre pour évaluer chacune des procédures qu'il contient et dans beaucoup de cas renvoyer exactement l'expression de départ. Par exemple si une expression contient la sous-expression $\exp(3/2)$, chaque étape du calcul va relancer la procédure `exp` qui va se rendre compte qu'elle ne sait rien faire d'autre sur $3/2$ que de renvoyer $\exp(3/2)$.

La stratégie de Maple face à cette question est la suivante :

- les variables `"`, `""`, et `"""` sont *toujours* évaluées totalement, ainsi que les arguments de la procédure `$` ;
- en utilisation interactive, l'évaluation totale est lancée à chaque étape ;
- dans une procédure, les expressions ne sont évaluées qu'à un niveau (sauf les exceptions déjà mentionnées) ;
- l'appel explicite à la commande `eval` permet de contrôler l'évaluation.

En pratique, cette stratégie a trois conséquences pour l'utilisateur. Lorsque l'on utilise `subs` et qu'il est important que les simplifications des fonctions aient lieu avant la suite du calcul, il faut alors faire `eval(subs(.))`. Deuxièmement, il faut éviter d'utiliser les variables locales `"`, `""` et `"""` dans les procédures lorsqu'elles représentent autre chose que des nombres, puisqu'alors on a une perte d'efficacité due à l'évaluation totale. Troisièmement, il faut éviter d'utiliser la procédure `$` le plus souvent possible ; il vaut mieux recourir à `seq`.

Pour le reste, il est important de bien comprendre la distinction entre variable et valeur mais les subtilités de l'évaluation à un ou plusieurs niveaux

n'ont quasiment aucune incidence sur la vie quotidienne de l'utilisateur de Maple.

EXEMPLE 45. Voici un exemple extrême :

```
f:=proc() local x,y; x:=y; y:=0; x end:
f();
```

$$y$$

Ici le dernier $x$ n'est évalué qu'à un niveau, et la procédure renvoie la variable $y$. (En revanche, si l'on demande ensuite la valeur de $y$, on obtient 0.)

EXEMPLE 46. Le cas le plus réaliste est celui des expressions dont l'arbre n'est pas parcouru à chaque étape du calcul. L'exemple suivant met en évidence ce phénomène :

```
s:=0: t:=proc(x) global s; s:=s+1; 't(x)' end:
f:=subs(x=t(3),(x^2-1)/(x-1)): s;
```

$$1$$

```
f,1+f,sin(f+z): s;
```

$$7$$

```
normal(eval(f,1)); s;
```

$$t(3)+1$$
$$7$$

```
normal(f); s;
```

$$t(3)+1$$
$$9$$

À chaque appel de la procédure `t`, le compteur `s` est incrémenté. L'expression `f` est une fraction rationnelle en `t(3)`. Sa création exige une évaluation de `t` lors de l'évaluation du premier argument de `subs`. Ensuite, chaque fois que `f` apparaît dans une expression au niveau interactif, son évaluation totale est déclenchée, ce qui incrémente `s` deux fois. L'évaluation à un niveau de `f` ne lance pas l'évaluation de `t`, pas plus que `normal`, puisqu'à l'intérieur de `normal`, toutes les évaluations ont lieu à un seul niveau. Le dernier test provoque l'évaluation de `t` puisque `normal`, comme presque toutes les procédures Maple, évalue totalement ses arguments. En revanche, si l'instruction `normal(f)` est utilisée à l'intérieur d'une procédure, elle ne déclenche pas l'évaluation de `t`, car la variable `f` n'est évaluée qu'à un seul niveau.

### 3.1.4. *Tables et procédures*

Trois particularités des tables et des procédures méritent d'être signalées : ces deux structures ne s'évaluent pas selon les mêmes règles que les autres objets du système, elles sont toutes deux susceptibles de création implicite et la commande `op` ne permet pas de les décomposer complètement.

*Création implicite.* La création implicite intervient lorsqu'une valeur de fonction ou une variable indexée apparaissent en membre gauche d'une affectation.

EXEMPLE 47. Les deux affectations

```
a[3]:=0: f(0):=1:
```

créent respectivement une table et une procédure.

La plupart des objets Maple ne peuvent pas être modifiés. Les opérations courantes créent une copie modifiée de leur argument. Il n'y a qu'une exception à cette règle : les tables. L'affectation à une variable indexée comme `a[3]` n'a de sens que si `a` est une table. Lorsque la variable `a` n'a pas été déclarée explicitement par `a:=table()`, le système, au lieu de protester en rencontrant l'affectation `a[3]:=0`, décide de créer une table et d'associer la valeur 0 à l'entrée 3.

Le cas des procédures est lié à l'utilisation de l'affectation pour remplir la table de *remember* (voir §1.3). Le système crée dans ce cas automatiquement une table de *remember* et une procédure qui ne fait rien sauf pour la valeur particulière qui a été définie.

```
print(f);
proc() options remember; 'procname(args)' end
```

*Règle d'évaluation.* La deuxième particularité des tables (donc des matrices qui sont des tables particulières) et des procédures est leur règle d'évaluation : lorsqu'une table ou une procédure a été affectée à une variable, *l'évaluation de celle-ci produit son nom.* Pour accéder au contenu de la table ou de la procédure, il faut explicitement faire appel à `eval`. En termes de flèches, l'évaluateur s'arrête de suivre les flèches lorsqu'il détecte un de ces objets.

Voici un exemple qui illustre une des conséquences de cette règle pour l'utilisateur

```
a:=3: b:=a: a:=2:
f:=proc() 0 end: g:=f: f:=2:
b,g;
```

$$3, 2$$

Dans le premier cas, au moment de l'affectation `b:=a`, `a` est évalué, et l'interprète affecte à `b` la valeur 3. Le fait que `a` soit modifié par la suite n'a aucune incidence sur `b`. L'interprétation en termes de flèches est donnée en figure 4 p. 61. Dans le second cas, au moment de l'affectation `g:=f`, `f` s'évalue en son nom, et donc la valeur affectée à `g` est, non pas la procédure, mais la variable `f`. Si `f` est modifié ensuite, la valeur de `g` l'est aussi. L'interprétation en termes de flèches est donnée en figure 5.

Une autre conséquence immédiate est que pour afficher la procédure ou la table, il faut appeler explicitement une des procédures `print` ou `eval`.

EXEMPLE 48.

```
print(a);
```


FIGURE 4 Évaluation des noms.

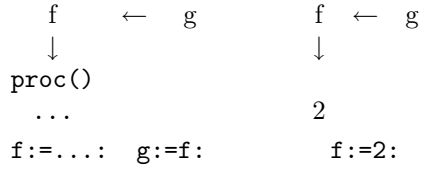



FIGURE 5 Évaluation des procédures.

table([
$1 = 3$
])

Cette règle spéciale d'évaluation est motivée par la possibilité de faire référence à des valeurs d'une procédure ou des entrées d'une table qui ne sont pas affectées. Ainsi `f(x)` ou `t[x]` sont affichées ainsi, au lieu que le corps de la procédure ou de la table apparaisse.

Pour les procédures, l'exemple que nous avons donné est extrême et la règle d'évaluation des procédures n'a pas d'autre effet apparent pour l'utilisateur que la nécessité d'utiliser `print` pour en afficher le corps. Cette règle est par ailleurs utile au bon fonctionnement du système. D'une part l'option *remember* de la procédure `readlib` ne fonctionnerait pas sinon : il faut en effet que l'argument de `readlib` soit le même d'un appel sur l'autre, que la procédure ait déjà été chargée ou non. D'autre part cela permet d'appeler la commande `trace` sans protéger l'argument.

Les tables sont les seuls objets Maple qui peuvent être modifiés. Cette modification s'effectue par l'affectation d'une valeur à une de leurs entrées. La commande `map` applique récursivement une procédure aux entrées d'une *copie* de la table qui lui est passée en argument. Ainsi la procédure `copy` est écrite comme un `map` de l'identité. Par commodité, il n'est pas nécessaire d'utiliser `map` pour appliquer `subs` ou `evalf` à une table.

Les deux conséquences les plus importantes de cette règle concernent l'algèbre linéaire, où la plupart des programmes, comme l'élimination gaussienne, modifient certains éléments des matrices passées en argument. Voici un prototype simplifié d'une telle procédure d'algèbre linéaire :

```
f:=proc(mat)
local b;
  b:=mat;
```

```
    b[2,1]:=b[2,1]−3*b[1,1];
    b
  end:
```

L'utilisation de cette procédure ne produit pas l'effet désiré :

```
a:=array([[1,2],[3,4]]);
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
f(a):
print(a);
```

$$\begin{bmatrix} 1 & 2 \\ 0 & 4 \end{bmatrix}$$

La matrice donnée en argument a été modifiée, car la variable `b` désigne physiquement la même matrice que `mat`, et donc que `a`. Il faut utiliser explicitement `copy`. La procédure s'écrit alors :

```
g:=proc(mat)
local b;
  b:=copy(mat);
  b[2,1]:=b[2,1]−3*b[1,1];
  b
end:
```

Mais l'utilisation de cette procédure réserve encore une surprise :

```
g(a);
```

$$b$$

Le résultat de la procédure est la variable locale `b`. Pour éviter les confusions que peut entraîner cette variable locale dans les calculs, il vaudra mieux écrire la procédure ainsi :

```
h:=proc(mat)
local b;
  b:=copy(mat);
  b[2,1]:=b[2,1]−3*b[1,1];
  eval(b)
end:
```

De cette façon, la procédure renvoie bien une matrice, et ne modifie pas son argument.

*Champs des tables et des procédures.* La dernière particularité des tables et des procédures concerne l'emploi des commandes `op`, `subsop` et `subs`. Les deux premières vont de pair : si un champ d'une structure peut être atteint par `op`, il est possible d'utiliser `subsop` pour créer une copie de la structure dont ce champ est modifié.

Pour les tables, la commande `op` joue un rôle différent de son rôle habituel : elle construit une liste des correspondances (entrée=sortie) de la table. Ceci pourrait laisser croire que les tables sont stockées comme des listes, mais il

TABLE 2 Les champs d'une procédure.

| Instruction | valeur |
|---|---|
| eval(f) | la procédure |
| op(0,eval(f)) | le type `procedure` |
| op(1,eval(f)) | ses paramètres |
| op(2,eval(f)) | ses variables locales |
| op(3,eval(f)) | ses options |
| op(4,eval(f)) | sa table de *remember* |
| op(5,eval(f)) | sa description (inexploitée) |
| op(6,eval(f)) | ses variables globales |

n'en est rien (voir §3.2). Les deux autres commandes d'accès aux tables sont `indices` qui construit la liste des entrées (clés) de la table et `entries` qui construit la liste des valeurs.

Les champs des procédures sont indiqués au tableau 2. Les remplacements de paramètres ou de variables locales par `subsop` doivent maintenir la cohérence entre les deux suites de noms. Il est par ailleurs impossible de remplacer la table de *remember* par une autre table. Le seul remplacement possible est par `NULL`, ce qui revient à vider la table.

EXEMPLE 49. Une version simple de la procédure `forget`, qui vide une table de *remember*, s'écrit ainsi :

```
easy_forget:=proc(p:procedure)
   p:=subsop(4=NULL,op(p))
end:
```

Une autre fonctionnalité de la procédure `forget` est la suppression d'une entrée particulière de la table. Bien que la commande `subsop` ne permette pas de remplacer la table de *remember* par une autre, cette suppression est rendue aisée par le fait que les tables ne sont pas recopiées :

```
forget_value:=proc(p:procedure,x)
local t;
   t:=op(4,op(p));
   t[x]:='t[x]';
   NULL
end:
```

Cette technique est en outre bien plus efficace, puisque contrairement à `subsop` elle n'entraîne pas la création d'une copie de la procédure. La procédure `forget` du système l'emploie d'ailleurs même pour vider toute la table.

Bien que la commande `op` ne permette pas d'avoir accès aux instructions d'une procédure, la commande `subs` s'y applique. Appliquée aux tables, la commande `subs` n'affecte pas les indices mais uniquement les valeurs.

Exemple 50. Nous créons une procédure :

```
p:=proc()
local a,b,c;
global x,f;
    a:=3;
    x:=a;
    f:=sin(x+b)
end:
```

puis nous y effectuons une substitution :

```
subs([a=s,x=t],sin=cos,eval(p));
```

proc() local s,b,c; global t,f; s := 3; t := s; f := cos(t+b) end

L'opération de substitution dans les procédures sert principalement à la création de procédures à partir d'un programme.

Exemple 51. La situation suivante se produit souvent : au cours d'un calcul il devient souhaitable de créer une procédure qui tienne compte de valeurs obtenues. L'approche naïve ne fonctionne pas :

```
a:=exp(x)+x+x^3+1:
b:=sin(x):
f:= x->if x>1 then a*b*ln(x) else a*b^2*ln(x) fi;
f(3);
```

$$\left(e^x + x + x^3 + 1\right) \sin(x) \ln(3)$$

Ce résultat où $x$ n'est pas partout remplacé par 3 s'explique par le moment où les variables `a` et `b` sont évaluées. Elles ne le sont pas à la création de la procédure, mais à son exécution. Pour obtenir le résultat souhaité, il faut utiliser dans la procédure non pas les variables mais leur valeur. La commande `subs` est alors nécessaire :

```
g:=subs('a'=a,'b'=b,eval(f)):
g(3);
```

$$\left(e^3 + 31\right) \sin(3) \ln(3)$$

Il est à noter que dans le cas où l'on veut créer une procédure directement à partir d'une expression, il est plus simple d'utiliser la fonction `unapply`. Ce n'est cependant pas possible dans l'exemple précédent à cause du test `if`.

Ce mécanisme de substitution est utile pour créer des procédures simples à l'intérieur d'un programme. Lorsque les procédures à créer sont plus complexes, il devient malaisé de les produire par de simples substitutions. Cette technique est alors remplacée avantageusement par le mécanisme plus général fourni par la procédure `procmake`.

*Options des procédures.* Les options d'une procédure ne modifient pas sa fonctionnalité, mais elles influent sur l'affichage ou l'exécution des calculs. Toutes les options sont valides, mais seules certaines sont reconnues par le système ; celles-ci sont indiquées au tableau 3.

TABLE 3 Options reconnues par le système.

| Nom | Rôle |
|---|---|
| builtin | indique une procédure du noyau |
| trace | le détail de l'exécution est affiché |
| remember | la table de *remember* est systématiquement remplie |
| Copyright... | le corps de la procédure n'est pas affiché |
| system | la table de *remember* est vidée périodiquement |
| operator,arrow | la procédure utilise la notation `a->f(a)` |
| operator,angle | la procédure utilise la notation `<f(a)\|a>` |
| package | reconnu par `with` |

L'option *builtin* caractérise les procédures du noyau, écrites en C. Par exemple

```
print(op);
proc() options builtin; 118 end
```

Environ 90 procédures font ainsi partie du noyau, le tableau 4 recense les principales d'entre elles.

Plus un programme utilise des procédures du noyau à la place de celles de la bibliothèque, plus il a des chances d'être efficace, c'est pourquoi il est bon d'avoir cette liste en tête. La plupart des commandes qui y sont mentionnées sont décrites dans ce livre, et toutes sont décrites dans la documentation en ligne.

Le reste du système Maple (soit environ 800 procédures) est entièrement écrit dans le langage Maple. Ces procédures sont protégées par l'option *copyright*. Leur affichage est réduit :

```
print(sin);
proc(x) ... end
```

Cette option est utilisable dans n'importe quelle procédure :

```
f := proc(n) option ‘Copyright toto‘; n! end;
f := proc(n) ... end
```

Pour afficher effectivement le code, il suffit de changer un des paramètres de l'interface, en exécutant l'instruction magique :

```
interface(verboseproc=2);
```

### 3.1.5. *Macros*

La commande `macro` permet de définir une abréviation. Ses arguments ne sont pas évalués. Nous avons vu que les variables utilisées à l'intérieur des procédures ne sont évaluées qu'à leur exécution et non à leur création. Les macros en revanche sont substituées dans les procédures au moment de leur création.

EXEMPLE 52.

TABLE 4 Fonctions du noyau.

| Domaine d'application | Nom des fonctions |
|---|---|
| éléments du langage | `ERROR`, `RETURN`, `macro`, `map`, `seq`, `type`, `traperror` |
| chaînes de caractères | `cat, substring, lexorder` |
| entiers, rationnels, flottants | `abs` , `igcd`, `iquo`, `irem`, `isqrt`, `max`, `min`, `modp`, `modp1`, `mods`, `evalf/hypergeom`, `trunc` |
| polynômes, fractions rationnelles | `coeff`, `coeffs`, `degree`, `denom`, `divide`, `icontent`, `lcoeff`, `ldegree`, `maxnorm`, `normal`, `numer`, `sign`, `sort`, `tcoeff` |
| fonctions | `diff`, `expand`, `series` |
| expressions | `assigned`, `convert`, `eval`, `evalb`, `evalf`, `evalhf`, `evaln`, `frontend`, `has`, `hastype`, `indets`, `length`, `nops`, `op`, `subs`, `subsop`, `$` |
| tables | `entries`, `indices`, `table` |
| ensembles, listes | `intersect`, `member`, `minus`, `union` |
| *package hackware* | `addressof`, `assemble`, `disassemble`, `pointto` |
| interface | `alias`, `appendto`, `lprint`, `print`, `userinfo`, `writeto`, `time`, `system`, `readlib` |

```
macro(f=‘nom de fonction/difficile a_taper‘);
g:=proc() f(x)+2 end;
   g := proc() ‘nom de fonction/difficile a_taper‘(x)+2 end
```

3.1.6. *L'interprète*

Outre le simplificateur et l'évaluateur dont nous avons beaucoup parlé dans cette section, l'interprète comprend une interface et un analyseur syntaxique. Cet analyseur est accessible via les commandes `parse` et `readline`. L'interprète peut être sommairement décrit comme effectuant la boucle infinie

```
do
    print(eval(parse(readline())))
od
```

où `readline` lit une chaîne de caractères, `parse` y reconnaît une instruction Maple, qui est évaluée par `eval`, et le résultat est finalement affiché par `print`. La commande `parse` nous sera utile pour calculer des temps d'exécution par la procédure

duree:=**proc**(r) time(eval(parse(r,statement))) **end**:

qui prend une chaîne de caractères en argument.

Exemple 53.

```
duree(‘for i to 50 do ifactor(2^i-1) od;‘);
                                1.033
```

Au niveau de l'interface, on dispose de la commande `alias` qui permet d'utiliser des abréviations. Contrairement à `macro`, ces abréviations n'affectent pas les expressions ou les procédures, mais uniquement leur affichage. Au démarrage du système, le nombre imaginaire $i$ est défini comme un alias de $(-1)^{1/2}$, ce qui permet la simplification automatique $i^2 = -1$.

**3.2. Structure interne des objets Maple.** Une grande partie de l'efficacité du système Maple lui vient d'une caractéristique spécifique : les objets y sont uniques. Nous allons décrire brièvement dans cette section le procédé utilisé et surtout ses conséquences pour l'utilisateur en quête d'efficacité.

*Hachage et simplification automatique.* Le *hachage* est une technique informatique permettant de créer des tables (clé,valeur) auxquelles l'accès s'effectue en temps constant (contrairement aux listes par exemple, où l'accès demande un temps proportionnel au nombre d'éléments de la liste).

Les tables en Maple sont des tables de hachage. Il s'ensuit en particulier que le temps de recherche dans la table de *remember* d'une procédure est indépendant du nombre d'entrées qui y ont été stockées.

Tous les objets utilisés au cours d'une session Maple sont stockés dans une table de hachage particulière. Ceci s'applique non seulement aux objets nommés, mais aussi aux expressions et sous-expressions qui interviennent durant le calcul. L'accès à cette table est fait de telle manière que sommes et produits sont référencés à une entrée qui ne dépend pas de l'ordre des opérandes. À la création d'une expression l'arbre représentant l'expression est construit du bas vers le haut. Les feuilles sont les premiers objets reconnus, puis les expressions bâties sur les feuilles,... Au fur et à mesure de cette création, les objets sont tous comparés à la table de hachage. Si le système s'aperçoit lors de cette comparaison que l'objet qu'il teste est nouveau, alors l'objet est créé et devient nœud de l'arbre. Si au contraire le système reconnaît l'objet comme ayant déjà été utilisé, l'objet n'est pas créé et c'est l'ancien objet qui devient nœud de l'arbre. Ceci garantit l'existence d'une seule copie de chaque expression en mémoire.

Grâce à cette table, le test d'égalité de deux expressions est effectué en temps constant et la simplification automatique des sommes, des produits ou des ensembles est effectuée très rapidement par tri des opérandes.

*Graphes acycliques dirigés.* Une conséquence de l'unicité des objets est que les arbres que nous avons utilisés pour représenter les expressions sont stockés comme des graphes acycliques dirigés (abrégés à l'anglaise DAGs). Un exemple de DAG est donné en figure 6 p. 68.

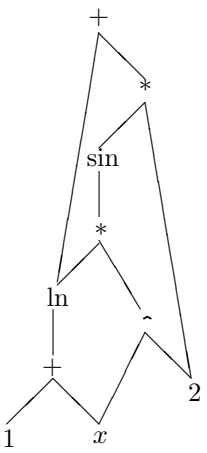



FIGURE 6 Le DAG représentant $\ln(x+1) + 2\sin(x^2 \ln(x+1))$.

Les conséquences de cette représentation sur l'espace mémoire utilisé sont énormes. Par exemple, la taille de la dérivée d'une expression de taille $n$ est de l'ordre de $n^{3/2}$ dans la représentation arborescente et est abaissée à l'ordre $n$ par l'utilisation de DAGs.

Il est plus délicat d'exploiter cette structure pour abaisser également le temps de calcul. L'option *remember* répond dans une certaine mesure à cette attente, mais au prix d'une consommation mémoire élevée. Nous décrivons une autre approche utilisant la commande `optimize` dans l'exercice 6.

*Package hackware.* Le *package hackware* est très délicat d'emploi et réservé au programmeur Maple confirmé. Cependant, il permet de comprendre vraiment comment fonctionne Maple et nous conseillons l'usage des commandes `disassemble` et `pointto` pour explorer la structure des objets Maple.

**3.3. Développement d'extensions du système.** Le langage de programmation permet d'étendre le système. Nous distinguons trois directions principales dans lesquelles l'utilisateur peut étendre Maple. La première est l'introduction dans le langage d'une fonction (au sens mathématique) qui n'est pas connue de Maple, la deuxième est l'ajout d'une fonctionnalité (une opération sur les fonctions mathématiques comme la dérivation ou l'évaluation numérique), et la troisième est l'écriture d'un ensemble de procédures formant une application particulière. Il y a dans tous les cas une philosophie générale de programmation qu'il faut conserver, et que nous allons détailler maintenant.

*Ajout d'une fonctionnalité.* Le petit programme de dérivation ci-dessous illustre une partie des points que nous avons présentés jusqu'ici. Cette procédure

prend deux arguments, l'expression à dériver et la variable par rapport à laquelle on dérive.

```
derive:=proc(f,x)
local s, i;
  if not has(f,x) then 0 # expressions indépendantes de x
  elif f=x then 1        # x lui-même
  elif type(f,§'+') then map(derive,f,x) # [ ∑ f_i]' = ∑ f_i'
  elif type(f,§'*') then # le produit est n-aire
                         # et [∏ f_i]' = ∑ f_i' ∏_{j≠i} f_i
    convert([seq(subsop(i=derive(op(i,f),x),f),
      i=1..nops(f))],§'+')
  elif type(f,§'^') then
    if not has(op(2,f),x) then                  # (f^a)' = a f' f^{a-1}
      op(2,f)*derive(op(1,f),x)*op(1,f)^(op(2,f)-1)
    else derive(exp(op(2,f)*ln(op(1,f))),x)     # f^g = exp(g ln f)
    fi
  elif type(f,function) then
    if op(0,f)=exp then f*derive(op(f),x)       # (e^f)' = f' e^f
    elif op(0,f)=ln then derive(op(f),x)/op(f)  # [ln(f)]' = f'/f
    else ERROR(§'Derivee non programmee')# on abandonne
    fi
  else ERROR(§'arguments invalides :'.f,x)
  fi
end;
```

Ainsi nous avons ajouté la fonctionnalité `derive` au système. Ce programme est simple, mais n'est pas très extensible : pour introduire une règle de dérivation pour une autre fonction, par exemple la tangente, il faut modifier la procédure. Dans la section suivante nous montrerons comment écrire cette procédure pour gérer de manière élégante de nouvelles fonctions.

Pour progresser en programmation Maple, nous suggérons au lecteur débutant de s'inspirer des programmes Maple existants. Nous avons déjà vu page 65 comment voir le contenu des fonctions de la bibliothèque. Pour écrire une procédure, une bonne méthode consiste à étudier une fonction existante qui effectue une tâche similaire. Le lecteur se forgera ensuite son propre style de programmation, et n'aura plus à regarder les fonctions du système que pour chercher à y repérer l'origine d'un dysfonctionnement.

*Ajout d'une fonction.* Supposons par exemple que la fonction logarithme n'existe pas en Maple. Ajouter la fonction ln ne signifie pas écrire une procédure `ln`. Ainsi, les fonctions d'Airy Ai et Bi sont connues de Maple, mais il n'existe pas de procédure correspondante. En effet, tant que les objets restent formels, la procédure `ln` se contente de renvoyer un résultat non

TABLE 5 Principales fonctions extensibles.

| | | |
|---|---|---|
| abs | evalf | Re |
| conjugate | expand | series |
| convert | Im | signum |
| diff | latex | simplify |
| evalapply | mod | type |
| evalc | print | value |

évalué. Le seul rôle d'une telle procédure est d'effectuer quelques simplifications triviales comme $\ln(1) \to 0$. En revanche, il est nécessaire d'étendre le système pour qu'il puisse dériver, simplifier, évaluer numériquement les expressions contenant `ln`. Autrement dit, une fonction mathématique n'est pas définie par une procédure, mais par plusieurs procédures correspondant aux propriétés de la fonction qui sont intéressantes pour les applications en vue. Dans le cas du logarithme, on écrira par exemple :

§‘diff/ln‘:=**proc**(y,x) diff(y,x)/y **end**:

ce qui traduit la relation habituelle :

$$[\ln(y)]' = \frac{y'}{y}.$$

C'est réellement ainsi qu'est définie la dérivée du logarithme en Maple. On écrira de même les autres procédures : `‘expand/ln‘`, `‘simplify/ln‘`, `‘evalf/ln‘`,... Dans chacun des cas, il suffit de se référer au `help` de la propriété que l'on souhaite préciser (`expand`, `simplify`,...) pour connaître le format des entrées et des sorties de la procédure que l'on écrit. Les principales procédures Maple extensibles de cette manière sont indiquées au tableau 5 p. 70.

Le fonctionnement de ce mécanisme est très simple. Nous le décrivons sur la procédure `derive` écrite plus haut, dont nous remplaçons la branche `elif` correspondant aux fonctions par

```
elif type(f,function) then
  fct:=‘derive/‘ . op(0,f);
  if not type(fct,procedure) then traperror(readlib(fct)) fi;
  if not type(fct,procedure) then
    ERROR(§‘Derivee non programmee : ‘. op(0,f))
  else fct(op(f),x)
  fi
```

Dans le cas d'une fonction `f`, le programme commence par tester l'existence d'une procédure `derive/f`. Si cette procédure n'existe pas, le programme tente alors de la trouver en bibliothèque. Si elle existe alors elle est appelée, sinon `derive` renvoie un message d'erreur. Il suffit d'écrire les procédures `‘derive/.‘` pour étendre les “connaissances” de la procédure `derive`, sans changer le corps de celle-ci.

*Ajout d'un* package *et organisation des fichiers.* Quelques règles sont à respecter lorsque l'on écrit un ensemble de procédures en Maple pour éviter que celles-ci n'interfèrent avec le reste du système. Ces règles visent essentiellement à éviter que ne soient redéfinies par erreur des variables ou des procédures. En général, il est impossible de redéfinir des procédures ou variables du système, car celles-ci sont protégées par `protect`, mais il faut éviter le plus possible d'introduire de nouveaux noms (même en les protégeant) que l'utilisateur risque d'utiliser comme variables (`x`, `y`, `sys`, `solution`, ...)

Toutes les variables globales en Maple (sauf celles qui sont accessibles à l'utilisateur comme `D`, `E`, `I`) sont précédées du caractère "`_`". Ainsi, la variable globale `_X` est constamment utilisée par la procédure `solve`, la variable `_Z` est utilisée par la procédure `RootOf`, les variables `_C1`, `_C2`,... sont utilisées par la procédure `dsolve` et l'affectation de l'une quelconque de ces variables perturbe fortement le fonctionnement du système. Lorsque des variables globales sont nécessaires, il faut leur donner un nom suffisamment long, peu susceptible d'être employé par l'utilisateur.

Quant aux procédures, nous distinguerons deux catégories : celles que l'utilisateur doit appeler et celles qui ne sont utilisées que par le programme.

Lorsque les premières sont en petit nombre, il suffit de leur choisir un nom suffisamment explicite, qui ne risque pas d'être déjà utilisé pour autre chose. Si ce n'est pas le cas, il faut créer un *package*, c'est-à-dire une table de procédures. Ainsi, si l'on a écrit trois procédures `proc1`, `proc2` et `proc3`, et que l'on décide d'appeler le *package* `toto`, on les renommera `toto[proc1]`,... par exemple en mettant la ligne

**macro**(proc1=toto[proc1],proc2=toto[proc2],proc3=toto[proc3]);

au début de chacun des fichiers où ces procédures sont utilisées. L'utilisateur pourra à son gré utiliser la forme longue des noms de procédures, ou faire appel à la commande `with(toto)`. Cette commande affecte à chaque nom court (par exemple `toto1`) la valeur correspondante de la table `toto`. Ensuite, si la procédure `toto/init` existe, `with` l'exécute. Cette procédure peut être utilisée pour affecter des variables globales nécessaires à l'utilisation du *package*.

Quant aux procédures que l'on ne souhaite pas rendre accessibles à l'utilisateur, on les nommera de préférence `toto/nom_de_procedure`. Cette utilisation éventuellement répétée du caractère "`/`" permet à la procédure `readlib` d'aller rechercher le fichier `nom_de_procedure.m` dans le répertoire ou dossier `toto` sur la plupart des systèmes d'exploitation (à condition que la variable `libname` soit correctement positionnée). Les gros développements sont donc structurés en plusieurs fichiers répartis dans des répertoires dont les noms sont ceux des procédures concernées. Dans ce cas, il peut également être utile de définir des procédures qui ne seront chargées qu'en cas de besoin, en utilisant un `readlib` protégé comme nous l'avons déjà vu. Enfin, sur certaines architectures, un programme indépendant de Maple appelé `march` permet de regrouper tous ces fichiers en un seul (M*aple* arch*ive*), dont `readlib` charge des morceaux à la demande.

Pour terminer de rendre le *package* utilisable, il faut écrire un fichier de description, reprenant les rubriques des messages habituellement fournis par `help`. La procédure `makehelp` transforme ce fichier en une procédure `help/text/toto` (ou `help/toto/text/toto1`) que la procédure `help` prend en compte.

*Outils de développement.* Pour comprendre pourquoi une procédure Maple — qu'elle fasse partie du système ou qu'on en soit l'auteur — ne fait pas ce que l'on en attend, on dispose de trois outils principaux : la variable globale `printlevel`, la commande `trace` et la variable `infolevel` couplée à la procédure `userinfo`.

La procédure `userinfo` s'utilise lors de l'écriture de programmes. Elle permet d'afficher des messages durant l'exécution du calcul. Par défaut, aucun message n'apparaît, l'affichage étant contrôlé par la table `infolevel`. Une partie des bibliothèques Maple fait usage de cette possibilité et le système affiche des messages après l'exécution de

```
infolevel[all]:=5:
```

La variable `printlevel` permet de suivre le déroulement des calculs, mais sans pouvoir spécifier la fonction ou le groupe de fonctions à observer. Sa valeur fixe le niveau de détail d'affichage de tous les calculs intermédiaires. Par exemple, après avoir défini

```
f:=proc(n)
   if n=1 then 1
   elif n mod 2 = 0 then f(iquo(n,2))
   else f(3*n+1) fi
end:
```

la commande `f(7)` affiche uniquement le résultat lorsque `printlevel` vaut 1. En augmentant cette valeur, on verra les calculs intermédiaires avec de plus en plus de détails. En outre, si le programme s'arrête avec une erreur (provoquée par `ERROR` et que `printlevel` vaut au moins 2, l'instruction qui a provoqué l'erreur est affichée, ainsi que la valeur de toutes les variables locales au moment de l'erreur.

Lorsque `printlevel` devient très élevé (de l'ordre de 100), il n'est pas toujours facile de traiter la masse énorme d'informations affichées. Quand c'est possible, il est plus informatif de ne faire afficher que ce qui concerne une ou deux procédures par la commande `trace`.

### 3.4. Exercices.

1. Reprendre l'exercice 3 de la page 44 en remplaçant la liste par un `array`.
2. Reprendre les questions de l'exercice 1 p. 54 en utilisant des tables pour stocker les listes. Les tables contiennent le champ `first` et le champ `suite` qui est à son tour une table ou `NULL` pour indiquer la fin de liste. On demande d'écrire deux variantes des procédures `append`, `cons`, `endcons`,

`reverse` et `rest` : une qui modifie la liste passée en argument et une autre qui renvoie une nouvelle liste.

3. Étendre Maple par l'ajout d'une fonction `invGAMMA` qui représente l'inverse de la fonction $\Gamma$ pour la composition. Écrire les procédures nécessaires à l'évaluation numérique, à la simplification et à la dérivation. [On pourra utiliser `solve`].
4. Étendre le système par l'ajout d'une procédure `lndiff` qui calcule la dérivée logarithmique d'une fonction (sans passer par `diff`). Cette procédure devra savoir traiter les fractions rationnelles, les fonctions ln, exp.
5. Poursuivre l'exercice précédent en définissant `lndiff` de telle façon qu'elle puisse être étendue par l'ajout de procédures `` `lndiff/toto` ``. Écrire alors `` `lndiff/sin` `` et `` `lndiff/cos` ``.
6. De nombreux évaluateurs (`series`, `evalf`, `evala`,..) établissent un morphisme entre deux types d'expressions : les sommes sont transformées en sommes, les produits en produits,... Dans ces circonstances, un gain d'efficacité important résulte d'une bonne utilisation des DAGs. Tous les évaluateurs de Maple emploient l'option *remember* à cette fin, mais cette utilisation a pour effet pervers de remplir inutilement la mémoire et d'empêcher ainsi l'exécution de gros calculs. Nous proposons dans cet exercice une autre approche plus efficace basée sur la commande `optimize`.
   (1) Utiliser la commande `optimize/makeproc` pour construire une procédure évaluant
   $$2x^2 \sin(x \ln(1+x)) + \sin(x^3 + x \sin(x \ln(1+x))) \ ;$$
   (2) remplacer dans le résultat de `optimize` les appels aux fonctions `sin` et `ln` par des appels à `series/sin` et `series/ln` avant d'appeler `optimize/makeproc` ;
   (3) en faisant `series` sur le résultat de la procédure obtenue à l'étape précédente vérifier que le résultat obtenu est le même que celui que l'on obtient en appelant directement `series` ;
   (4) automatiser ce calcul de séries.

# Seconde partie

# Domaines d'utilisation et applications

CHAPTER III

# Courbes et surfaces

La visualisation de données ou de résultats facilite la perception de phénomènes mathématiques ou physiques. Tous les systèmes de calcul formel offrent des fonctionnalités graphiques de base identiques : tracé à partir d'une représentation paramétrique, d'une équation ou d'une liste de points, utilisation de la souris pour changer le point de vue d'un tracé en trois dimensions, pour modifier l'aspect par des menus, pour effectuer des sorties en vue d'une impression,... Cependant, contrairement aux domaines plus théoriques où les commandes sont contraintes par la pratique mathématique, les applications graphiques ont des syntaxes très différentes d'un système à l'autre. Ainsi, le contenu de ce chapitre est plus spécifiquement lié au système Maple que les autres chapitres de cette partie.

Les tracés graphiques de ce chapitre ont été obtenus en utilisant le mode haute résolution de Maple. C'est pratiquement toujours le mode par défaut, sauf sur machine Unix quand on lance le système par la commande `maple` : il faut alors faire `plotsetup(x11)` pour avoir des tracés utilisant X Window. On suppose dans certaines parties de ce chapitre que le *package* `plots` de Maple a été chargé.

## 1. Tracés en deux dimensions

Le tracé le plus simple est celui d'une courbe dans un espace à deux dimensions. Cette courbe peut être définie sous au moins quatre formes différentes : une fonction en coordonnées cartésiennes donnée sous la forme $y = f(x)$ ou $x = f(t), y = g(t)$, une fonction implicite $f(x, y) = 0$, ou une fonction en coordonnées polaires $r = f(\theta)$.

**1.1. Courbes $y = f(x)$.** Commençons par l'exemple simple d'une fonction définie et continue sur un intervalle borné. En Maple, deux syntaxes sont possibles selon que la fonction à tracer est une procédure ou une expression. Dans le premier cas, on écrit `plot(f,a..b)` où `f` est une procédure renvoyant un nombre réel ; dans le second cas, on écrit `plot(e,x=a..b)` où `e` est une expression dépendant uniquement de la variable $x$ (fig. 1).

1.1.1. *Reproduction d'un graphique*

Lorsque Maple est utilisé dans un système de fenêtrage, la sauvegarde et l'impression de graphiques s'effectuent à la souris. Pour sauvegarder un

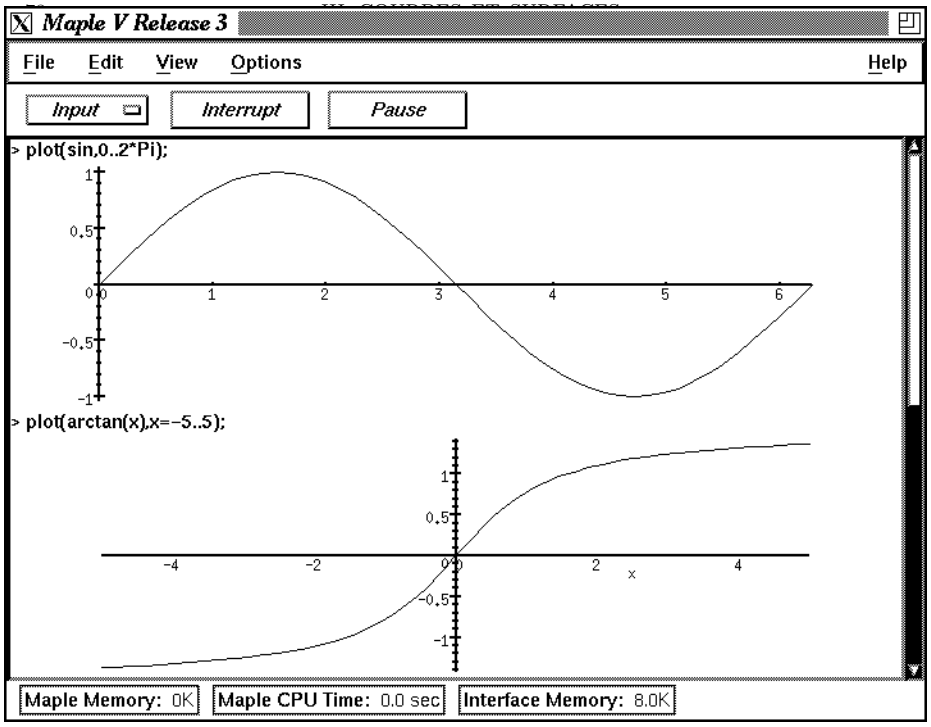



FIGURE 1 Deux tracés simples.

graphique en format *postscript*, on sélectionne dans le menu intitulé *File* le sous-menu *Print*, puis l'option *Postscript* (fig. 2).

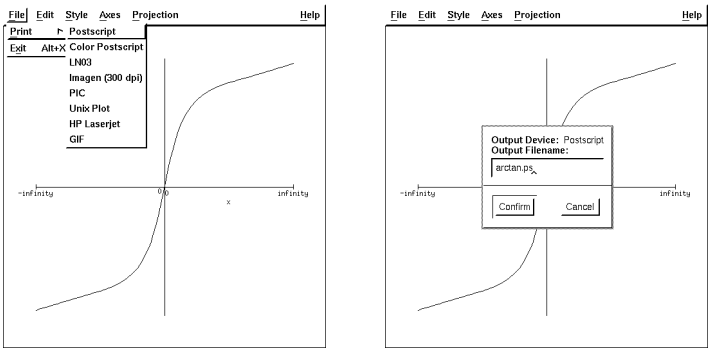
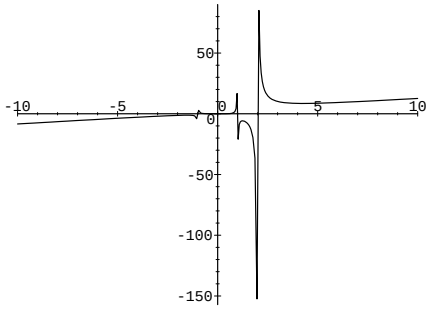


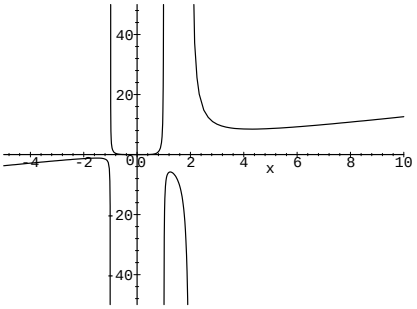

FIGURE 2 Exemple de sortie *postscript* d'un tracé. À gauche on sélectionne *Print* puis *Postscript* dans le menu *File*. À droite on choisit le nom du fichier et on lance sa création.

Pour inclure un graphique dans un *worksheet* Maple, comme en figure 1, on commence par le copier (option *Copy* du menu *Edit*), puis on se positionne à l'endroit voulu du *worksheet* et on l'insère avec l'option *Paste*.




```
plot(x^4/(2-x-2*x^2+x^3),
       x=-10..10);
```

FIGURE 3




```
plot(x^4/(2-x-2*x^2+x^3),x=-5
..10,y=-50..50,discont=true);
```

FIGURE 4

Dans la plupart des cas, la commande de tracé (`plot`) appelée directement donne un résultat satisfaisant. Cependant il est utile d'apprendre à reconnaître les "mauvais tracés", et de savoir alors comment aider le système.

EXEMPLE 1. Considérons la fonction $f(x) = x^4/(2-x-2x^2+x^3)$. Si l'on exécute `plot(x^4/(2-x-2*x^2+x^3),x=-10..10)`, on obtient le tracé de la figure 3. Au vu de ce tracé, on pourrait croire que la fonction $f$ est continue, et bornée entre $-150$ et 100, alors qu'elle a en fait trois pôles en $-1$, 1 et 2. D'autre part on distingue mal le comportement au voisinage de 0. Pour parer à ces problèmes, il faut limiter l'intervalle de variation des abscisses à celui qui nous intéresse, par exemple $[-5, 10]$, limiter l'intervalle de variation des ordonnées, par exemple $[-50, 50]$, et indiquer à la commande `plot` par l'option `discont=true` de chercher les discontinuités de la fonction. On obtient alors le tracé de la figure 4.

*Fonctions à variation rapide.* Lorsque l'on trace une fonction à variation rapide, par exemple une fonction oscillante comme $\sin(1/x)$ au voisinage de zéro, le système calcule un plus grand nombre de points que par défaut (49 en Maple) afin d'obtenir un rendu lisse. Pour s'en convaincre, il suffit d'ajouter l'option `style=point` qui trace les points calculés sans les relier. La figure 5 p. 80 montre un tel tracé contenant 69 points dans la première moitié $[0, 1/2]$ et 24 dans la seconde $[1/2, 1]$. Ce mécanisme de raffinement automatique est très évolué en Maple et il est difficile de trouver une fonction non nulle et continûment dérivable sur $[0, 1]$ dont le tracé (fait d'un nombre fini de points) se confond avec celui de la fonction nulle.

*Bornes infinies.* Dans le cas de bornes infinies, Maple utilise une transformation ramenant à l'intervalle fini $[-1, 1]$ ($x/8$ pour $|x| \leq 4$ et $(|x| - 2)/x$ pour $|x| > 4$). Ceci autorise la visualisation de fonctions sur des intervalles

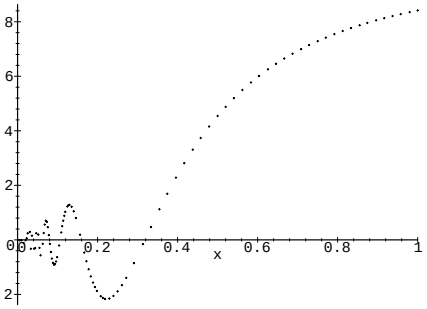



FIGURE 5 `plot(x*sin(1/x), x=0..1,style=point);`

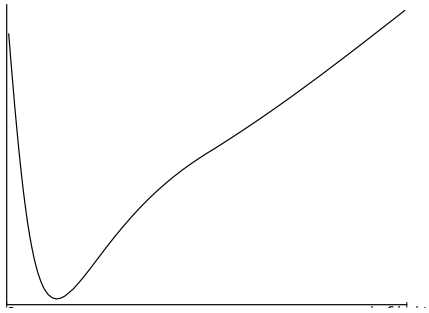


FIGURE 6 `plot(arctan(x+1/x), x=0..infinity);`

de largeur infinie en donnant simplement `infinity` ou `-infinity` pour l'une des bornes (fig. 6).

*Recherche de zéros.* La commande `fsolve` a parfois du mal à détecter les racines d'équations du type $f(x) = 0$. On peut l'aider en déterminant à l'aide d'un graphique un intervalle dans lequel se trouvent des racines.

EXEMPLE 2. Cherchons les racines de l'équation $\sin^2 x - \exp(-x)\cos x = 0$ dans l'intervalle $[0, 10]$. Le graphe de la figure 7 montre une racine dans l'intervalle $[0.4, 0.8]$, une ou deux autres dans les intervalles $[6, 6.4]$ et $[9.2, 9.6]$. Le tracé de la fonction sur $[6, 6.4]$ montre qu'il y a effectivement deux racines dans cet intervalle ; par contre le tracé sur $[9.2, 9.6]$ ne permet pas de conclure. Il faut restreindre le tracé à l'intervalle $[9.41, 9.44]$ pour pouvoir conclure qu'il n'y a pas de racine entre 9 et 10 (fig. 8).

**1.2. Courbes paramétriques et polaires.** Les tracés de courbes paramétriques $x = f(t), y = g(t)$ sont réalisés par `plot([f(t),g(t),t=a..b])` où $[a, b]$ est l'intervalle de variation du paramètre (fig. 9). Les tracés de courbes

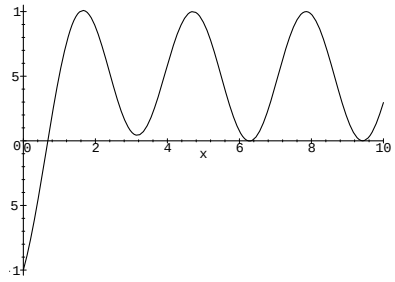



FIGURE 7 `plot(sin(x)^2-exp(-x)*cos(x),x=0..10);`

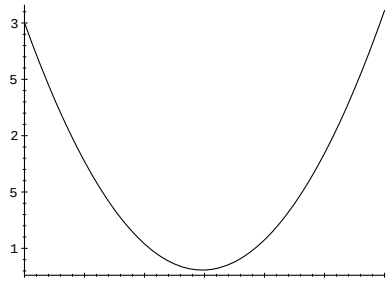



FIGURE 8 `plot(sin(x)^2-exp(-x)*cos(x),x=9.41..9.44);`

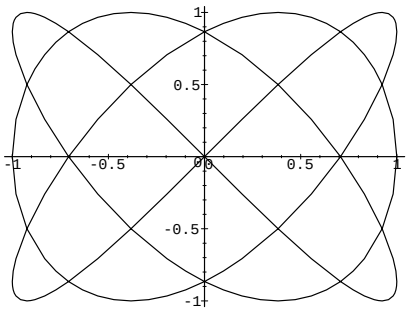



FIGURE 9 `plot([sin(3*t), sin(4*t),t=0..2*Pi]);`

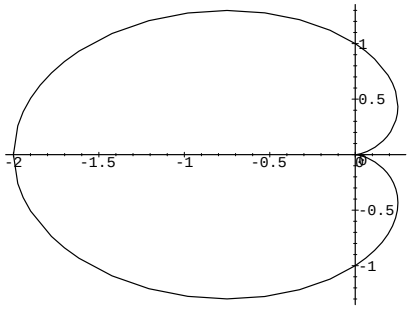



FIGURE 10 `polarplot(1-cos(t),t=0..2*Pi);`

définies en coordonnées polaires se font avec la commande `polarplot` du *package* `plots`, à laquelle on peut donner une expression ou une fonction pour une définition de la forme $r = f(\theta)$, ou bien une liste `[f(t),g(t),t=a..b]` pour $r = f(t), \theta = g(t)$ (fig. 10).

**1.3. Courbes implicites.** Il est également possible de tracer des courbes définies de façon implicite par une équation $f(x, y) = 0$, à l'aide de la commande `implicitplot` (fig. 11). On a cependant intérêt à trouver une représentation paramétrique (à l'aide de la commande `solve` ou d'un changement de variable astucieux) pour obtenir un meilleur rendu, surtout au voisinage des points multiples comme $(0, 0)$ dans notre exemple (comparer les figures 11 et 12).

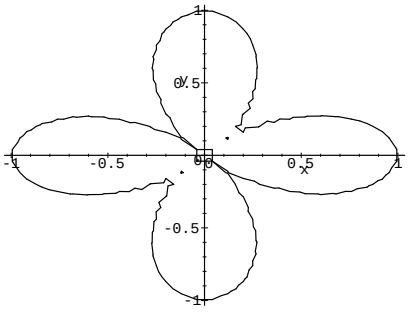



FIGURE 11 `implicitplot(x^6+ y^6+3*x^4*y^2+3*x^2*y^4-x^4+2* x^2*y^2-y^4,x=-1..1,y=-1..1);`

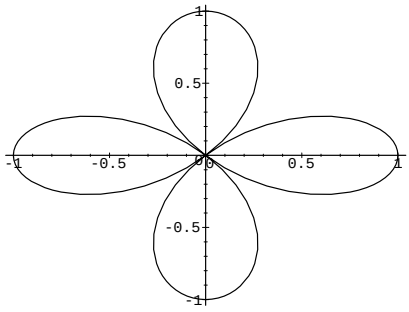


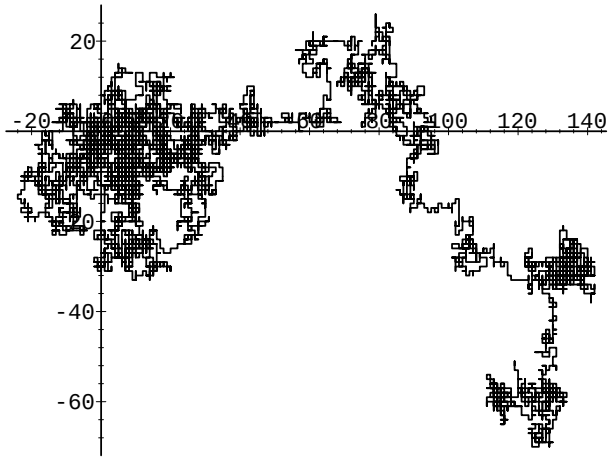

FIGURE 12 `polarplot(cos(2*t),t=0..2*Pi);`

**1.4. Tracé de données.** Pour tracer simplement une liste de points, en les reliant ou non, on utilise la commande `plot(l)` où `l` est la liste en question

(formée soit de couples `[x,y]`, soit de nombres, les indices impairs correspondant alors aux abscisses et les indices pairs aux ordonnées).

EXEMPLE 3. Voici une marche aléatoire dans le plan. Partant du point $(0, 0)$, un point mobile fait des pas nord, sud, est, ouest de façon équiprobable.

```
x:=0: y:=0: dx:=[1,-1,0,0]: dy:=[0,0,1,-1]: die:=rand(1..4):
for i to 10000 do l[i]:=x,y;r:=die();x:=x+dx[r];y:=y+dy[r] od:
plot([seq(l[i],i=1..10000)]);
```




*Tracé de données issues d'un fichier.* Lorsque les données à visualiser sont obtenues en dehors du système, par une expérience physique ou un autre programme, la solution la plus simple pour visualiser ces données en Maple consiste à passer par l'intermédiaire d'un fichier. On peut alors soit formater ce fichier à l'extérieur du système, en ajoutant des virgules et des crochets au bon endroit, soit utiliser la commande `readdata` de Maple (voir l'aide en ligne). Avec `readdata`, l'utilisateur n'a plus à modifier le fichier de résultats, ce qui permet de renouveler le tracé pour autant d'ensembles de résultats qu'il le désire.

EXEMPLE 4. Le programme C suivant calcule la somme partielle $h_i = 1 + 1/2 + \cdots + 1/i$ et $\log(i)$ pour $i$ de 1 jusqu'à 100.

```
main(){ double i=1.0,s=0.0;
   for (;i<=100;s+=1/i,i+=1) printf("%f %f %f\n",i,s,log(i));
}
```

L'exécution de ce programme produit cent lignes comme

```
74.000000 4.874509 4.304065
```

Si ces lignes sont envoyées dans le fichier `resultats`, on fera comme suit pour obtenir la liste des triplets $[i, h_i, \log i]$.

```
l:=readdata(resultats,3);
```

On se sert alors de la commande `seq` pour extraire les quantités désirées, par exemple pour faire un tracé de $h_i$ en fonction de $i$

```
extrait:= (i,j) -> [seq([x[i],x[j]],x=l)]:
plot(extrait(1,2));
```

ou un tracé comparatif de $h_i$ et $\log i$ en fonction de $i$ (les tracés simultanés sont réalisés via des ensembles, comme on le verra au §3.1).

```
plot({extrait(1,2),extrait(1,3)});
```

### 1.5. Exercices.

1. À l'aide de la commande `plot`, rechercher graphiquement les racines réelles du polynôme $x^5+3x^4+x^2+8x+3$ ; vérifier avec la commande `fsolve`.
2. Tracer la courbe $r = 1-\cos\theta\sin\theta$ en coordonnées polaires. Inclure le graphique dans la session Maple et l'imprimer en Postscript (si votre plateforme le permet).
3. Soit la courbe du plan définie par $x^3y+xy^4-3x^2y-2x^2y^3+x^3+3y^4=0$. Tracer cette courbe dans le domaine $-1 \leq x, y \leq 3$ [Indication : utiliser l'option `numpoints` pour affiner le tracé.]
4. Tracer en fonction de $n$ le nombre de décimales correctes de $\pi$ obtenues en substituant $1/2$ à $x$ dans le développement limité de $6\arcsin x$ à l'ordre $2n$, pour $n = 1$ jusqu'à 100.
5. L'ensemble de Mandelbrot (fig. 13 p. 84). Étant donné un complexe $c = a+ib$, on définit la suite $(z_n)$ par $z_0 = 0$ et $z_{n+1} = z_n^2 + c$ pour $n \geq 0$. L'ensemble de Mandelbrot est formé par les points $(a, b)$ du plan tels que la suite associée $(z_n)$ reste à distance finie de l'origine. Une condition suffisante pour que $c$ ne soit pas dans l'ensemble est qu'il existe $n$ tel que $|z_n| < |z_n|^2 - |c|$.
   (1) Écrire une procédure prenant comme argument $n$ et calculant la valeur de $|z_n|^2 - |z_n| - |c|$ en fonction de $a$ et $b$.
   (2) À l'aide d'une procédure de tracé de fonctions implicites, dessiner les courbes obtenues pour $n = 1, 2, 3, 4$. En les superposant, on doit obtenir le graphique de la figure 13 p. 84. Poursuivre en augmentant les valeurs de $n$.
6. Le pavage de Rice. On considère le pavage du plan par des pentagones dont une partie est dessinée en figure 14 p. 84. Ce pavage a été découvert par Marjorie Rice en 1976.
   (1) Écrire les équations reliant les cinq angles et les cinq longueurs.
   (2) Les résoudre à l'aide d'un système de calcul formel.
   (3) Montrer que le système possède un degré de liberté, plus précisément que pour chaque valeur de l'angle aigu $\gamma$ du pentagone telle que
   $$2\arcsin\frac{\sqrt{3}-1}{2} \leq \gamma \leq 2\arcsin\frac{\sqrt{13}-1}{4}$$
   il existe une solution. Les valeurs extrêmes correspondent à des pavages par des quadrilatères.

(4) Reproduire le tracé de la figure 14 obtenu avec $\gamma = 0.9$.
(5) Observer la déformation du pavage lorsque $\gamma$ varie. Pour cela on pourra utiliser l'option `insequence=true` de la commande `display` avec une séquence de tracés obtenus pour différentes valeurs de $\gamma$ (voir §3.5).

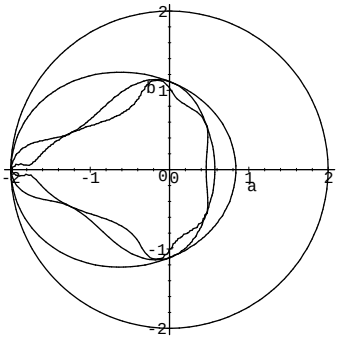



FIGURE 13 Ensemble de MANDELBROT.

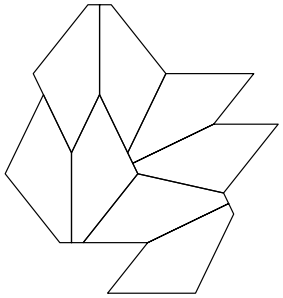


FIGURE 14 Pavage de RICE.

## 2. Tracés en trois dimensions

Les possibilités offertes par les systèmes sont les mêmes qu'en dimension deux, avec la différence qu'il y a maintenant deux types d'objets : les courbes et les surfaces. Les courbes sont rendues comme en dimension deux par des points reliés ou non ; les surfaces sont rendues par des maillages rectangulaires ou triangulaires avec élimination des parties cachées.

**2.1. Surfaces $z = f(x, y)$.** L'écran ou la feuille de papier n'offrant que deux dimensions, les systèmes projettent la scène en considérant que l'observateur est placé en un point précis de l'espace appelé *point de vue*, éventuellement infiniment éloigné, ce qui revient à faire une projection orthogonale. La direction du point de vue est donnée en coordonnées sphériques et en degrés par l'option `orientation`=[$\theta$,$\varphi$], avec par défaut $[45, 45]$ (fig. 15). On peut aussi changer le point de vue avec la souris en "cliquant" sur le tracé et en le faisant tourner, ou encore obtenir un aspect de perspective avec l'option `projection=r` où $r$ est compris entre 0 et 1, la valeur par défaut 1 correspondant à la projection orthogonale.

**2.2. Courbes et surfaces paramétrées.** Les courbes paramétrées $x = f(t), y = g(t), z = h(t)$ sont tracées par la commande `spacecurve` (fig. 16 p. 85). Les surfaces paramétrées $x = f(t, u), y = g(t, u), z = h(t, u)$ sont dessinées par `plot3d([f(t,u),g(t,u),h(t,u)],t=a..b,u=c..d)`.
Deux cas particuliers sont les surfaces définies en coordonnées cylindriques

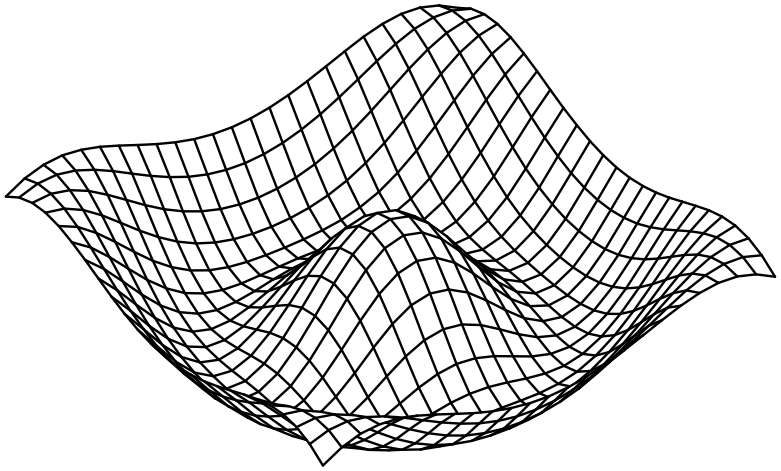
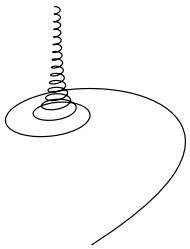
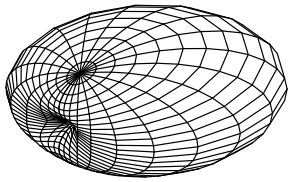


FIGURE 15
`plot3d(cos(sqrt(x^2+y^2)),x=-5..5,y=-5..5,orientation=[55,20]);`



`spacecurve([cos(t)/t,sin(t)/t,`
`t,t=1..100],numpoints=1000);`

FIGURE 16



`sphereplot(1-cos(t)*sin(p),`
`t=0..2*Pi,p=0..Pi);`

FIGURE 17

— $x = r\cos\theta, y = r\sin\theta, z = z$ avec $r = f(\theta, z)$ — et sphériques — $x = r\cos\theta\sin\varphi, y = r\sin\theta\sin\varphi, z = r\cos\varphi$ avec $r = f(\theta, \varphi)$ — qui sont dessinées respectivement par `cylinderplot` et `sphereplot` (fig. 17).

**2.3. Surfaces implicites.** Comme pour les courbes en dimension deux, il est possible de tracer des surfaces définies par une équation $f(x, y, z) = 0$ à

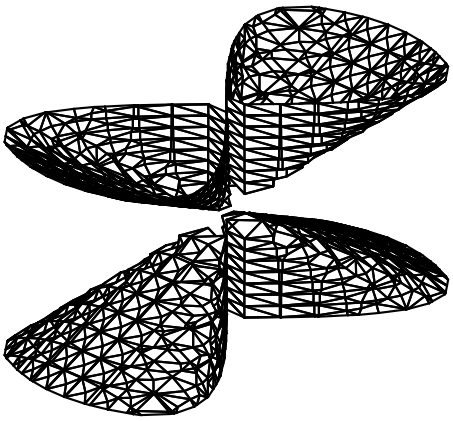


FIGURE 18 `implicitplot3d(x^4+ (y^2-x^2)*z^2,x=-1..1,y=-1..1, z=-1..1,grid=[20,20,20]);`

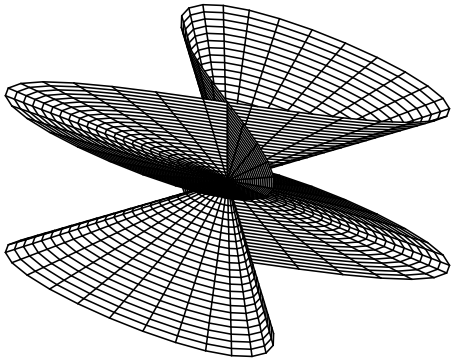
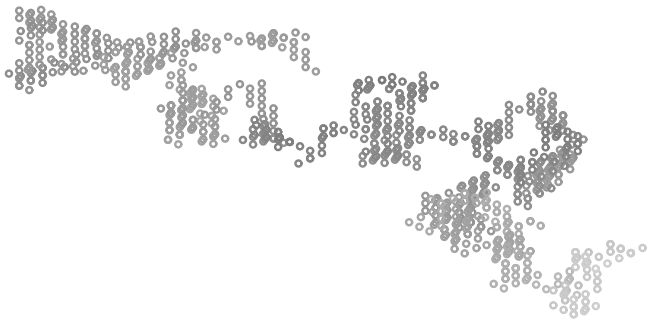


FIGURE 19 `plot3d([z*sin(t), z*cos(t)*sin(t),z],z=-1..1, t=0..2*Pi,grid=[50,50]);`

l'aide de la commande `implicitplot3d`. Un tel tracé est visible en figure 18 pour l'équation $x^4 + (y^2 - x^2)z^2 = 0$ dans la région $|x| \leq 1, |y| \leq 1, |z| \leq 1$. Comme en dimension deux, pour obtenir un meilleur rendu, il faut essayer de trouver une paramétrisation de la surface. Dans le cas d'une fonction homogène comme $x^4 + (y^2 - x^2)z^2$, on peut prendre l'une des coordonnées comme paramètre, soit $z$, et exprimer les autres à partir de celle-ci : $x = uz$ et $y = vz$. Ceci conduit à l'équation $u^4 + v^2 - u^2 = 0$, qui admet comme solution $u = \sin t, v = \cos t \sin t$. C'est ainsi qu'est obtenu le tracé de la figure 19, qui est bien meilleur, cinq fois moins coûteux en temps et quatre fois en mémoire.

**2.4. Tracé de données.** Pour tracer un ensemble de points en trois dimensions, on utilise la commande `pointplot` (si l'on veut que les points soient reliés entre eux, on aura recours à `spacecurve`). Reprenons l'exemple des marches aléatoires, qui peuvent modéliser la trajectoire d'une particule en dimension trois. Une telle marche est produite et dessinée de la façon suivante. (Sur un écran couleur, la troisième dimension est visualisée par la couleur, qui apparaît ici en niveaux de gris.)

```
x:=0: y:=0: z:=0: die:=rand(1..6):
dx:=[1,-1,0,0,0,0]: dy:=[0,0,1,-1,0,0]: dz:=[0,0,0,0,1,-1]:
for i to 1000
do l[i]:=x,y,z;r:=die();x:=x+dx[r];y:=y+dy[r]:z:=z+dz[r] od:
pointplot([seq([l[i]],i=1..1000)]);
```



Pour tracer un ensemble de points issus d'un fichier, on aura recours à la commande `readdata` (voir § 1.4).

**2.5. Tracé d'intersection.** Pour tracer l'intersection de deux surfaces dont les équations sont $f(x, y, z) = 0$ et $g(x, y, z) = 0$ (ce qui revient à tracer une courbe définie de façon implicite en trois dimensions), on pourra utiliser la commande `intersectplot` de la *share library*, que l'on charge par

```
with(share); readshare(intplot, plots);
```

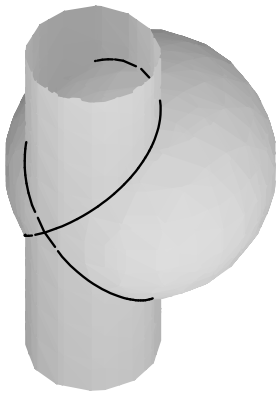
La figure 20 p. 88 montre le tracé simultané d'un cylindre, d'une sphère et de leur intersection.

**2.6. Exercices.**

1. Vérifier graphiquement que les surfaces
$$z = y^2 - x^2 \quad \text{et} \quad z = 2/3 + \cos x - \cos y$$
n'ont pas d'intersection sur $-1 \leq x, y \leq 1$, en modifiant le point de vue avec la souris, puis le prouver formellement.
2. Dessiner la surface paramétrée définie par
$$x = (2 - v\sin(u/2))\sin u, \quad y = (2 - v\sin(u/2))\cos u, \quad z = v\cos(u/2)$$
sur le domaine $0 \leq u \leq 2\pi$, $-1 \leq v \leq 1$.
3. Dessiner la surface définie par
$$-4x + x^3 + xy^2 + 2x^2z - 4zy + 2zy^2 + z^2x = 0$$
dans le domaine $-3 \leq x \leq 3$, $-3 \leq y \leq 3$, $-1 \leq z \leq 1$.

## 3. Autres types de tracés

**3.1. Tracés simultanés.** Les tracés simultanés d'objets de même type sont réalisés en donnant un ensemble de fonctions ou d'expressions à la commande graphique (fig. 21). En revanche, lorsque les objets sont de types différents (une surface et une courbe, ou une surface en coordonnées cylindriques et une autre en coordonnées sphériques), il faut procéder de la manière suivante. On commence par calculer les différents tracés et on affecte les résultats à des variables intermédiaires. On utilise ensuite une des commandes `display` ou `display3d` selon que les objets sont bi ou tri-dimensionnels (fig. 20).



```
c:=implicitplot3d(x^2+y^2-x,x=-1..1,y=-1..1,z=-1.5..1.5):
s:=implicitplot3d(x^2+y^2+z^2-1,x=-1..1,y=-1..1,z=-1..1):
i:=intersectplot(x^2+y^2+z^2-1,x^2+y^2-x,[x,y,z],color=black):
display3d({c,s,i},style=patchnogrid,scaling=constrained);
```

FIGURE 20 Intersection d'une sphère et d'un cylindre.

**3.2. Lignes de niveau.** Une fois tracée une surface, on peut ajouter les lignes de niveau à la souris, avec le menu *Style*. Il est également possible de les obtenir en vue de dessus avec la commande `contourplot` (fig. 22).

On obtient le même résultat avec la commande `plot3d` et les options `style=contour` et `orientation=[-90,0]`. On peut aussi avoir à la fois les lignes de niveau et un rendu de la surface avec l'option `style=patchcontour`, comme représenté sur la figure 23 pour la fonction

$$f = \exp(-x^2 - y^2) + 2\exp(-(x-3)^2 - y^2) + 3\exp(-(x-1)^2 - (y-2)^2).$$

**3.3. Tracé point par point.** Pour certaines applications, le tracé de points non reliés apporte une information supplémentaire. Par exemple cela permet de visualiser l'accélération d'un point mobile. En Maple on utilise pour ce faire l'option `style=point` avec la commande `plot` en dimension deux ou `spacecurve` en dimension trois (fig. 24).

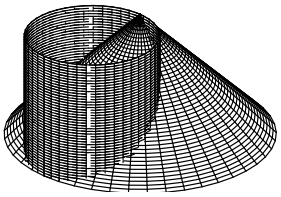


FIGURE 21

```
cylinderplot({2*cos(t),1-z},t=0..2*Pi,z=-1..1,numpoints=2000);
```

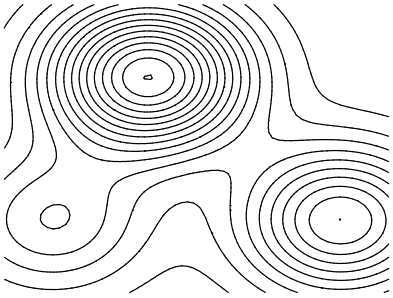


FIGURE 22 `contourplot(f, x=-1/2..7/2,y=-1..3, numpoints=4000);`

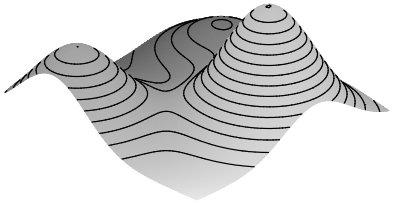
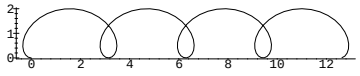
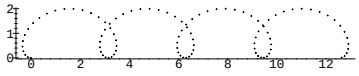


FIGURE 23 `plot3d(f, x=-1/2..7/2,y=-1..3,numpoints= 4000,style=patchcontour);`




`plot(l,axes=framed);`




`plot(l,axes=framed,style=point);`

FIGURE 24 Tracé continu et tracé de points.
`l=[seq([t/8-sin(t/4),1-cos(t/4)],t=0..100)],scaling=constrained`

**3.4. Tracés en couleur.** La couleur apporte également une dimension de plus, ce qui permet de visualiser des objets en quatre dimensions. Une application typique est le tracé d'une fonction du plan complexe dans lui-même. Une telle fonction a pour argument un nombre complexe $z = x + iy$, et retourne un autre nombre complexe $z' = x' + iy'$. On peut alors représenter en trois dimensions $x'$ en fonction de $x$ et $y$, la couleur caractérisant $y'$. Si $f$ est l'expression de la fonction à tracer, on fait comme suit :

```
plot(Re(f),x=a..b,y=c..d,color=Im(f));
```

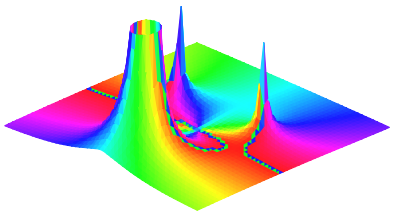
Pour visualiser le module de $z'$ en fonction de $x$ et $y$, la couleur représentant l'argument de $z'$, on remplace `Re(f)` par `abs(f)` et `Im(f)` par `argument(f)` (fig. 25).

**3.5. Animation.** L'animation permet également de visualiser une quatrième dimension, de nature plutôt temporelle. Elle consiste en l'affichage séquentiel de plusieurs tracés.

EXEMPLE 5. L'instruction

```
animate([t/4-r*sin(t/4),1-r*cos(t/4),r=0..1],t=1..100);
```

visualise la trajectoire d'un rayon de bicyclette.



```
g:=subs(z=x+I*y,1/(1-z^3)*log(1/(1-z))^2):
 plot3d(abs(g),x=-2..2,y=-2..2,view=0..8,
     color=argument(g),grid=[50,50]);
```

FIGURE 25 Représentation du module d'une fonction analytique avec l'argument en couleur (ici en niveaux de gris).

Le tracé ne s'exprime pas nécessairement par une fonction. L'affichage s'obtient alors par les commandes `display` et `display3d` avec une liste de tracés graphiques et l'option `insequence=true`.

La suite d'instructions ci-dessous montre l'évolution d'un groupe de dix particules partant d'un même point et se déplaçant aléatoirement au nord, au sud, à l'est ou à l'ouest, jusqu'à une centaine de déplacements.

```
N:=10: T:=100: x:=array(1..N,0..T): y:=array(1..N,0..T):
for i to N do x[i,0]:=0; y[i,0]:=0 od:
die:=rand(1..4): dx:=[1,-1,0,0]: dy:=[0,0,1,-1]:
for j to T do for i to N do r:=die();
   x[i,j]:=x[i,j-1]+dx[r]; y[i,j]:=y[i,j-1]+dy[r] od od:
for j from 0 to T
do p[j]:=plot([seq([x[i,j],y[i,j]],i=1..N)],style=point) od:
display([seq(p[j],j=0..T)],insequence=true,symbol=circle);
```

Maple ouvre alors une fenêtre (fig. 26) avec plusieurs boutons : le premier ■ stoppe une animation en cours, le second ■ ▶ lance une animation, le troisième ▶| avance d'un pas, les deux boutons suivants ▶▶ et ◀◀ augmentent et diminuent respectivement la vitesse d'animation, le bouton ⇌ fait passer en mode *auto-reverse* et le bouton → contrôle le sens de parcours.

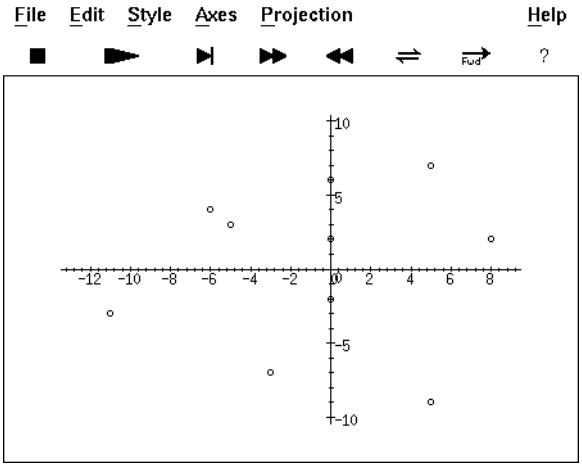



FIGURE 26 Animation en Maple.

CHAPTER IV

# Arithmétique et combinatoire

Les mathématiques discrètes sont omniprésentes dans un système de calcul formel. De façon apparente lors du calcul d'un pgcd d'entiers, ou à l'insu de l'utilisateur lors d'une factorisation de polynôme, le système manipule des nombres entiers. En Maple, c'est également le cas lors des calculs en nombres flottants, qui sont codés à partir des entiers. Il est donc important pour l'utilisateur d'apprécier le coût des principales opérations effectuées sur les entiers. Ces questions sont abordées en §1, ainsi que des techniques simples pour accélérer certains calculs. Nous montrons aussi dans cette section les équations dont un système de calcul formel sait ou devrait savoir trouver les solutions entières. La seconde partie de ce chapitre est consacrée à la combinatoire et plus précisément au dénombrement. Dans ce domaine, c'est l'aptitude des systèmes à manipuler des grands entiers de manière exacte qui est la plus utile, mais nous insisterons également sur l'utilisation de séries génératrices.

## 1. Arithmétique

**1.1. Calculs entiers et rationnels.** Dans un système de calcul formel, l'utilisateur n'a pas à se soucier des calculs entiers ou rationnels. Ils sont effectués automatiquement par le système, quelle que soit la taille des nombres manipulés. Toutefois, la connaissance de quelques principes de base permet d'accélérer les calculs.

Ainsi, il faut savoir qu'ajouter deux nombres de $n$ chiffres prend un temps proportionnel à $n$ ; les multiplier prend un temps plus long, proportionnel à $n^2$ sur la plupart des systèmes.

Exemple 1. La commande `duree` introduite au chapitre II permet de vérifier ces assertions :

```
a:=3^20959: b:=7^11832: aa:=a*a: bb:=b*b:
duree(‘to 1000 do a+b od‘),duree(‘to 1000 do aa+bb od‘);
```

$$0.683, 1.067$$

```
duree(‘a*b‘),duree(‘aa*bb‘);
```

$$0.367, 1.500$$

Ici $a$ et $b$ sont deux entiers de 10 000 chiffres, leurs carrés ont 20 000 chiffres ; le temps de calcul sur les carrés est le double pour l'addition et le quadruple pour le produit.

Dans la mesure du possible, il faut donc supprimer les produits au profit des additions, même si le nombre d'additions devient élevé.

Une autre façon de réduire la longueur des calculs consiste à éviter autant que possible d'avoir recours à des nombres rationnels. Dans les principaux systèmes de calcul formel, les nombres rationnels sont en effet automatiquement réduits : $6/4$ devient $3/2$ et $1/2 + 1/3$ devient $5/6$. Ces simplifications peuvent influer de façon notable sur le temps de calcul, notamment lorsque numérateur et dénominateur sont de grands entiers.

Exemple 2. Pour calculer la somme

$$\sum_{n=0}^{499} \frac{1}{n!},$$

la méthode naturelle consiste à opérer directement en ajoutant à chaque itération le rationnel $1/n!$ :

```
s:=1: t:=1: for n to 499 do t:=t/n; s:=s+t od:
```

mais il vaut mieux ajouter l'entier $500!/n!$ à chaque étape et diviser par $500!$ à la fin :

```
u:=0: t:=1: for n from 500 by -1 to 1 do t:=t*n; u:=u+t od:
u:=u/500!: evalb(s=u);
```

*true*

La première méthode est environ 60 fois plus lente que la seconde : à chaque itération, le système effectue une réduction au même dénominateur alors qu'il n'en fait qu'une seule dans le second cas.

### 1.1.1. *Exponentiation*

Qu'il s'agisse d'entiers, de résidus d'entiers modulo $p$, de polynômes ou de matrices, la multiplication est une opération relativement coûteuse. L'exponentiation — le calcul de $a^n$ où $n$ est un entier — est un cas important où le nombre de multiplications peut être réduit.

La méthode élémentaire consiste à multiplier par $a$ jusqu'à obtenir la puissance voulue. Elle nécessite $n-1$ multiplications. Il est facile de faire beaucoup mieux ; par exemple $a^{32} = ((((a^2)^2)^2)^2)^2$ et cette expression n'utilise que cinq multiplications. De façon générale, l'algorithme d'*exponentiation binaire* est le suivant :

$$a^n = \begin{cases} a & \text{si } n = 1, \\ a \cdot a^{n-1} & \text{si } n > 1 \text{ est impair}, \\ (a^{n/2})^2 & \text{si } n > 1 \text{ est pair}. \end{cases} \tag{1}$$

Cet algorithme n'effectue que des carrés et des multiplications par $a$. Si la représentation binaire de $n$ est $(b_k \dots b_1 b_0)$, le nombre de carrés est $k$ et le nombre de multiplications égale le nombre de $b_i$ égaux à 1 (sans compter $b_k$ qui vaut toujours 1). Autrement dit, le nombre d'opérations égale la longueur plus le nombre de 1 de la représentation binaire de $n$ après avoir enlevé le 1

de tête. Dans tous les cas, il est majoré par $2\log_2 n$, où $\log_2$ représente le logarithme en base 2.

Voici le programme récursif correspondant :

```
expo:=proc(a,n:posint)
    if n = 1 then a
    elif irem(n,2)=1 then a*expo(a,n−1)
    else expo(a,iquo(n,2))^2
    fi
end:
```

et en voici une version itérative :

```
expo:=proc(a,n:posint)
local l, res, f, i;
    f:=a;
    l:=convert(n,base,2);
    if l[1]=1 then res:=a else res:=1 fi;
    for i in subsop(1=NULL,l) do
            f:=f*f;
            if i=1 then res:=res*f fi
    od;
    res
end:
```

**1.2. Divisibilité et primalité.** Une grande partie de la richesse de l'arith_métique provient de la possibilité d'effectuer des divisions euclidiennes. Le calcul de pgcd en découle, d'où une forme réduite pour les fractions. Nous commencerons par ces questions avant de passer à une autre conséquence importante de la division euclidienne : l'existence d'une décomposition en facteurs premiers.

1.2.1. *Division euclidienne et pgcd*

Dans la plupart des systèmes de calcul formel, les rationnels sont automatiquement réduits. La réduction des rationnels s'effectue par des calculs de pgcd : $a/b$ est simplifié en $a'/b'$ avec $a' = a/\mathrm{pgcd}(a,b)$ et $b' = b/\mathrm{pgcd}(a,b)$.

Le temps de calcul de cette réduction est dominé par celui du pgcd. Pour s'en convaincre, il suffit de comparer les temps de calcul du pgcd et du quotient de deux grands entiers.

Exemple 3. Avec $a$ et $b$ les entiers de 10 000 chiffres définis p. 93,

```
duree(‘igcd(a,b)‘), duree(‘a/b‘);
```

7.866, 7.867

Un calcul de pgcd (`igcd`) a le même ordre de complexité qu'une multiplication, alors qu'une division d'entiers (`iquo` donne le quotient et `irem` le reste) a le même ordre de complexité qu'une addition.

Exemple 4. Toujours sur les mêmes nombres :

```
duree(‘to 1000 do iquo(a,b) od‘);
```
$$1.217$$

Du fait de la fréquence de ces opérations dans un système de calcul formel, les fonctions effectuant la division euclidienne de deux entiers ou calculant le pgcd d'entiers font partie du noyau et sont donc très rapides.

Le pgcd d'entiers n'est pas calculé par l'algorithme d'Euclide, qui est relativement lent. En revanche, l'algorithme d'Euclide étendu (`igcdex` en Maple) permet de calculer une combinaison linéaire donnant le pgcd :

```
g:=igcdex(9407449,4284179,'u','v'):
g,u,v;
```
$$541, 1690, -3711$$

Ceci donne une identité de Bézout : $541 = 1690 \cdot 9407449 - 3711 \cdot 4284179$.

En particulier, une telle identité permet de résoudre l'équation

$$ax \equiv b \bmod n.$$

Si $ua + vn = g$ où $g$ est le pgcd de $a$ et $n$, alors soit $b$ est un multiple de $g$ et la solution est donnée par $x = bu/g \bmod n$, soit il n'y a pas de solution.

1.2.2. *Théorème des restes chinois*

Le théorème des restes chinois permet de calculer un (grand) entier dont on connaît une majoration, et dont on sait calculer les résidus modulo d'autres nombres plus petits. Les calculs menés modulo des petits entiers sont bien sûr plus rapides que ceux sur des grands entiers. Les opérations sur les grands nombres sont ainsi limitées à la première étape, où le problème est codé modulo des petits entiers, et à la dernière, où le résultat est reconstruit (par `chrem`). Le principe consiste à calculer suffisamment de résidus modulo des nombres bien choisis pour reconstruire l'entier cherché.

Exemple 5. Pour calculer le produit de $a = 143563$ par $b = 718976$, prenons par exemple $p_1 = 9999$, $p_2 = 9998$ et $p_3 = 9997$, qui sont deux à deux premiers entre eux. Le produit $ab$ étant inférieur au produit des $p_i$, il peut être reconstruit à partir de ses restes modulo les $p_i$. Chacun de ces restes se calcule avec des entiers de même taille que $p_i$, en remarquant que $ab \bmod p_i = (a \bmod p_i)(b \bmod p_i) \bmod p_i$.

```
a:=143563: b:=718976:
l:=[9999,9998,9997]:
m := [seq((a mod p)*(b mod p) mod p,p=l)];
```
$$m := [4355, 9286, 6284]$$

```
chrem(m,l), a*b;
```
$$103218351488, 103218351488$$

Une application plus réaliste est le calcul d'un déterminant. La preuve de la nullité du déterminant d'une grande matrice par calcul direct peut faire apparaître des entiers intermédiaires très grands. Cette explosion des entiers intermédiaires est évitée par un calcul modulaire. La taille du résultat est facilement majorée en fonction de la taille de la matrice et de la plus

grande des entrées. Chaque calcul modulo $p$ doit renvoyer 0, et le résultat s'en déduit après avoir pris suffisamment d'entiers $p$. Dans ce contexte, la fonction `nextprime` est très utile.

### 1.2.3. *Primalité et factorisation*

Décider de la primalité d'un entier ou calculer sa décomposition en facteurs premiers sont des opérations bien plus coûteuses que celles que nous avons vues jusqu'ici.

EXEMPLE 6. Soit $p$ le produit des nombres premiers d'indice 1 000 à 1 050.

```
p:=convert([seq(ithprime(i),i=1000..1050)],`*`):
time(isprime(p)), time(ifactor(p));
```

$$0.633, 8.517$$

Ce nombre n'a que 200 chiffres. Pourtant sa décomposition en facteurs premiers est vingt fois plus coûteuse que la multiplication de deux entiers de 10 000 chiffres !

Le test de primalité de Maple, comme celui de la plupart des systèmes de calcul formel, est probabiliste. Lorsqu'il répond `false`, le nombre est à coup sûr composé ; par contre, lorsqu'il répond `true`, le nombre n'est que très probablement premier. Ceci étant dit, aucun contre-exemple n'est connu à ce jour (pour la version V.3).

La méthode probabiliste utilisée est une combinaison du test de FERMAT (si $p$ est premier, alors $a^p = a \bmod p$ pour tout entier $a$) et du test de LUCAS, dont l'explication sort du cadre de ce livre.

Pour le test de FERMAT avec $a = 2$, les contre-exemples inférieurs à 2 000 sont 341, 561, 645, 1 105, 1 387, 1 729, 1 905. Un nouveau test avec $a = 3$ ne laisse subsister que 561, 1 105 et 1 729. Ces trois entiers passent le test de FERMAT avec n'importe quelle base $a$ : ce sont des nombres de CARMICHAEL ; on sait depuis 1993 qu'il en existe une infinité. Par contre, on ne sait pas à l'heure actuelle s'il existe des entiers mettant en défaut à la fois le test de FERMAT et celui de LUCAS.

Le test de primalité probabiliste est très rapide. Il existe aussi des tests de primalité garantis, qui sont plus lents, mais ils ne sont pas disponibles en Maple.

La factorisation d'entiers est une opération beaucoup plus coûteuse. Sur les ordinateurs actuels, des programmes prouvent la primalité d'entiers d'un millier de chiffres. Pour la factorisation, la limite se situe vers une centaine de chiffres. Par exemple, la revue *Scientific American* publia en 1977 l'entier de 129 chiffres

$$114381625757888867669235779976146612010218296721242362562561842935706935245733897830597123563958705058989075147599290026879543541.$$

Sa factorisation n'a été trouvée qu'en 1994. Le dixième nombre de FERMAT $F_{10} = 2^{2^{10}} + 1$ fournit un autre exemple : il se décompose en $45592577 \times$

$6487031809 \times N$ où $N$ est un entier de 291 chiffres. Cet entier est composite, puisqu'il ne passe pas le test de FERMAT en base 3 :

```
N := (2^(2^10)+1)/45592577/6487031809:
evalb(Power(3,N) mod N = 3);
```

*false*

```
isprime(N);
```

*false*

mais on ne connaît toujours pas sa décomposition en facteurs premiers.

Il faut donc être très prudent avec les commandes `ifactor` et `ifactors` et éviter de les utiliser dans un programme si une autre solution est possible.

*Primalité et codes secrets.* La facilité relative avec laquelle on peut créer des nombres premiers, comparée à la difficulté de la factorisation d'un nombre composite, a donné lieu au système de code secret à clé publique RSA, du nom de ses inventeurs RIVEST, SHAMIR et ADLEMAN.

Le principe est le suivant : le destinataire du message choisit deux grands nombres premiers $p$ et $q$. Il calcule trois entiers : le produit $n = pq$, un entier $e$ premier avec $p-1$ et $q-1$, et l'inverse $d$ de $e$ modulo le produit $(p-1)(q-1)$ (l'algorithme d'EUCLIDE étendu appliqué à $e$ et $(p-1)(q-1)$ donne un moyen de calculer $d$, voir §1.2.1). Il donne alors $n$ et $e$ à l'expéditeur du message, en gardant $p$, $q$ et $d$ secrets.

Pour transmettre son message, l'expéditeur commence par le coder en entiers de $\{0, \ldots, n-1\}$ (par exemple en l'écrivant en binaire, puis en le coupant en morceaux). Pour communiquer un entier $a \in \{0, \ldots, n-1\}$, l'expéditeur transmet $b = a^e \bmod n$. Le destinataire retrouve le nombre $a$ car $a = b^d \bmod n$.

Seuls doivent être tenus secrets les entiers $p$, $q$ et $d$. Le nombre $n$ peut être rendu public, car sa seule connaissance ne permet pas le décodage. En effet, pour trouver $d$, il faut d'abord trouver $p-1$ et $q-1$, c'est-à-dire factoriser $n$.

Pour coder un texte, il faut commencer par le transformer en un entier[†] :

```
Alphabet:=§‘ abcdefghijklmnopqrstuvwxyz‘:
nb_lettres:=length(Alphabet):
for i to nb_lettres do lettre[i−1]:=substring(Alphabet,i..i) od:
for i to nb_lettres do codage[eval(lettre[i−1],1)]:=i−1 od:

transforme := proc(s:string) local i;
   convert([seq(codage[substring(s,i..i)]*nb_lettres^(i−1),
         i=1..length(s))],‘+‘)
end:
detransforme := proc(l:posint) local i;
```

[†]Noter l'utilisation de `eval` pour éviter les problèmes qui surgiraient si l'une des lettres était utilisée comme variable.

```
    cat(seq(eval(lettre[i],1),i=convert(l,base,nb_lettres)))
  end:
```

et voici les fonctions de codage proprement dites :

```
  encode := proc(a,e,n) local i;
    [seq(Power(i,e) mod n,i=convert(transforme(a),base,n))]
  end:
  decode := proc(b,d,n) local mess, i;
    mess:=map(Power,b,d) mod n;
    detransforme(convert([seq(mess[i]*n^(i-1),i=1..nops(mess))],`+`))
  end:
```

L'utilisation de RSA est extrêmement simple. Par exemple avec

$$p = 2324923849189, \qquad q = 433110035200049$$

et $e = 21466760385753970647708001 3$, qui est bien premier avec $(p-1)(q-1)$, Maple trouve facilement la valeur de $d$ lorsque $p$ et $q$ sont connus :

```
p := 2324923849189:  q := 433110035200049:
n := p*q:  e := 21466760385753970647708001 3:
d := 1/e mod (p-1)*(q-1);
```

$$d := 46964770317266328756124640 5$$

Les entiers $e$ et $n$ sont alors transmis à un ami. Il s'en servira pour encoder son message ainsi :

```
code := encode(`rendez vous ce soir`,e,n);
```

$$code := [3056414258810826698991580, 1]$$

Connaissant $d$, le décodage est aisé :

```
decode(code,d,n);
```

*rendez vous ce soir*

Le grand avantage du système RSA est que tout ce que l'expéditeur connaît (sauf le message) peut être divulgué, y compris les fonctions de transformation entre texte et entiers, et même le codage utilisé, d'où le nom de code "à clé publique".

**1.3. Fractions continues.** Les fractions continues fournissent de bonnes approximations des nombres réels par des rationnels. Étant donnés un entier $q$ et un réel positif $r$, l'entier $p$ le plus proche de $qr$ fournit une approximation $p/q$ différant d'au plus $1/(2q)$ de $r$. Si le choix de $q$ est libre, il existe des paires d'entiers $(p, q)$ pour lesquelles l'approximation est bien meilleure : $|r - p/q| \leq 1/q^2$. Le développement en fraction continue de $r$ donne une suite de telles approximations. Ce développement s'écrit

$$r = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots}}}.$$

Dans ce développement, $a_0$ est la partie entière de $r$, puis $a_1$ est la partie entière de $1/(r-a_0)$,... Les approximations rationnelles sont les *réduites du développement*, qui s'obtiennent en normalisant les résultats intermédiaires :

$$a_0, \quad a_0 + \frac{1}{a_1}, \quad a_0 + \frac{1}{a_1 + \dfrac{1}{a_2}}, \dots$$

Dans cette suite, les dénominateurs $q$ sont croissants. Cette suite est finie lorsque $r$ est rationnel et infinie sinon. La suite des $a_i$ est périodique si et seulement si $r$ est algébrique de degré 2 (racine d'un polynôme du second degré à coefficients entiers).

EXEMPLE 7. En Maple, le développement en fraction continue de `r` est donné par la commande `convert(r,confrac)`. Cette commande renvoie la liste des quotients partiels (les $a_i$) et met la liste des réduites dans un troisième argument optionnel.

```
convert(evalf(Pi,11),confrac,'l'): l;
```

$$\left[3, \frac{22}{7}, \frac{333}{106}, \frac{355}{113}, \frac{103993}{33102}, \frac{104348}{33215}, \frac{208341}{66317}, \frac{312689}{99532}, \frac{1459097}{464445}\right]$$

Il est à noter que la dernière réduite donnée par Maple ne doit pas être prise en compte ; elle est souvent fausse du fait de l'erreur numérique introduite par `evalf`.

Il existe aussi des fractions continues à termes polynomiaux. La fraction continue associée au développement limité $a_0 + a_1 x + \cdots$ est $F_0 = a_0 + x/F_1$ où $F_1$ est celle associée à $1/(a_1 + a_2 x + \cdots)$. Les réduites s'appellent des approximants de PADÉ diagonaux et sont utiles pour le calcul numérique.

EXEMPLE 8. Voici le début du développement de $\exp(x)$ :

```
convert(exp(x),confrac,x);
```

$$1 + \cfrac{x}{1 + \cfrac{x}{-2 + \cfrac{x}{-3 + \cfrac{x}{2 + \frac{1}{5}x}}}}$$

L'ordre du développement est réglé par la variable `Order`.

**1.4. Équations en nombres entiers.** Ces équations sont aussi appelées équations diophantiennes. Nous indiquons pour chacune des classes suivantes ce que le calcul formel peut apporter.

1.4.1. *Équations linéaires*

Dans le cas de deux variables, ce sont les équations de la forme

$$ax + by = c$$

où $a$, $b$ et $c$ sont des entiers, ainsi que les inconnues $x$ et $y$. Deux cas sont possibles : si $c$ n'est pas multiple du pgcd de $a$ et $b$, il n'y a aucune solution,

```
isolve(2*x+4*y=5);
```

sinon l'algorithme d'EUCLIDE étendu (voir p. 96) appliqué à $a$ et $b$ garantit l'existence d'une solution $(x_0, y_0)$. Il existe alors une infinité de solutions $(x_0 + kb, y_0 - ka)$ avec $k$ entier quelconque.

```
isolve(15*x-28*y=9);
```

$$\{y = 15\_N1 - 63, x = 28\_N1 - 117\}$$

Dans le cas d'une équation linéaire à trois variables ou plus, le même principe s'applique, et il y a soit zéro, soit une infinité de solutions.

```
isolve(7*x+3*y+13*z=1);
```

$$\{z = -\_N1 - 3\_N2 + 1, x = \_N1, y = 2\_N1 + 13\_N2 - 4\}$$

Dans le cas de plusieurs équations linéaires, il existe un changement de variables qui ramène le système sous forme triangulaire ou diagonale. Sans rentrer dans les détails, car tout cela est transparent à l'utilisateur via la procédure `isolve`, Maple sait résoudre tous les systèmes linéaires d'inconnues entières.

```
isolve({7*x+6*y+13*z=2,6*x+8*y+12*z=2});
```

$$\{y = 1 - 3\_N1, z = -3 + 10\_N1, x = 5 - 16\_N1\}$$

*Application au calcul de fréquences.* Les interférences entre des émissions radiophoniques se produisent quand leurs fréquences $f_1, \ldots, f_n$ vérifient une relation de la forme

$$\lambda_1 f_1 + \lambda_2 f_2 + \cdots + \lambda_n f_n = 0 \tag{2}$$

où les $\lambda_i$ sont des "petits" entiers non tous nuls. Les fréquences peuvent être considérées comme des entiers car elles sont toutes multiples d'une certaine fréquence de base.

Étant données les fréquences $f_1, \ldots, f_n$, l'équation (2) a une infinité de solutions. L'indice d'interférence d'une solution $(\lambda_1, \ldots, \lambda_n)$ est défini comme la somme des carrés des coefficients $\lambda_i$. Plus l'indice est petit, plus il y a interférence entre les différentes émissions.

Puisque l'équation est linéaire, `isolve` s'applique :

```
f1:=344: f2:=591: f3:=872:
s := isolve(a*f1+b*f2+c*f3=0);
```

$$s := \{a = -4\_N1 - 145\_N3, b = -8\_N1 - 72\_N3, c = 7\_N1 + 106\_N3\}$$

La somme des carrés des coefficients est par conséquent

```
e := expand(subs(s,a^2+b^2+c^2));
```

$$e := 129\_N1^2 + 3796\_N1\_N3 + 37445\_N3^2$$

Quelles que soient les valeurs des $f_i$, l'indice d'interférence obtenu est une forme quadratique homogène en $N_1, N_3, \ldots$ Par conséquent, son minimum est atteint en $N_1 = N_3 = \cdots = 0$. Comme ce point est exclu, il faut chercher le second minimum, atteint en un point voisin.

```
seq(subs(i,e),i=[seq(seq([_N1=j,_N3=k],j=-1..1),k=-1..1)]);
```

$$41370, 37445, 33778, 129, 0, 129, 33778, 37445, 41370$$

L'indice non nul minimal vaut donc 129. Ce calcul se généralise aisément en une procédure. Cette procédure peut ensuite être utilisée pour choisir des fréquences avec une interférence minimale.

1.4.2. *Équations quadratiques*

Ce sont les équations de degré total deux. Dans le cas de deux variables, Maple les résout de façon exacte. Dans certains cas, comme pour les équations de la forme $x^2 + y^2 = n$, il n'y a qu'un nombre fini de solutions.

Exemple 9.

```
isolve(x^2+y^2=97);
```

$$\{x = -4, y = -9\}, \{y = 9, x = 4\}, \{y = -9, x = 4\}, \{y = -4, x = -9\},$$
$$\{x = -4, y = 9\}, \{x = 9, y = 4\}, \{y = -4, x = 9\}, \{x = -9, y = 4\}$$

En revanche, les équation du type $x^2 - Dy^2 = 1$, appelées équations de Pell, ont une infinité de solutions en général.

Exemple 10.

```
s:=isolve(x^2-2*y^2=1): s[1];
```

$$\left\{ y = \frac{1}{4}\sqrt{2}\left(\left(3+2\sqrt{2}\right)^{\_N1} - \left(3-2\sqrt{2}\right)^{\_N1}\right), \right.$$
$$\left. x = \frac{1}{2}\left(3+2\sqrt{2}\right)^{\_N1} + \frac{1}{2}\left(3-2\sqrt{2}\right)^{\_N1} \right\}$$

Ici `isolve` retourne quatre solutions, correspondant aux deux signes possibles pour $x$ et pour $y$ ; nous avons sélectionné la première. Voici les six premières valeurs de cette solution :

```
seq(simplify(s[1]),_N1=0..5);
```

$$\{y = 0, x = 1\}, \{x = 3, y = 2\}, \{y = 12, x = 17\},$$
$$\{y = 70, x = 99\}, \{y = 408, x = 577\}, \{y = 2378, x = 3363\}$$

Dans le cas de trois variables ou plus, il n'y a pas de méthode générale. Maple sait traiter quelques équations particulières, comme l'équation de Pythagore, qui correspond aux triangles rectangles à côtés entiers (essayer `isolve` sur l'équation $x^2 + y^2 = z^2$). La commande `isolve` sait dénicher des solutions particulières,

```
isolve(x^2+y^2+z^2=0);
```

$$\{y = 0, x = 0, z = 0\}$$

mais la plupart du temps ne trouve aucune solution,

```
isolve(x^2+y^2+z^2=97);
```

ce qui ne signifie pas pour autant qu'il n'y en a pas (ici les triplets $(9, 4, 0)$ et $(6, 6, 5)$ sont des solutions). Il est dommage que le système ne fasse pas de différence entre "il n'y a pas de solution" et "je ne sais pas trouver les solutions".

*Décompositions de Lagrange.* Un théorème célèbre de LAGRANGE affirme que tout entier s'écrit comme somme de quatre carrés. L'équation $x^2+y^2+z^2+t^2=n$ a donc au moins une solution pour toute valeur de $n$. Comme pour $x^2+y^2=n$, le nombre de solutions est fini. Pour les trouver, la fonction `isolve` ne s'applique pas puisqu'il y a quatre variables. Cependant il n'est pas difficile d'écrire un programme les énumérant. Le nombre de solutions peut être déterminé sans passer par l'énumération, nous y reviendrons au §2.2.2.

### 1.4.3. *Équations elliptiques et congruences*

Les équations diophantiennes elliptiques sont de la forme $y^2=P(x)$ où $P$ est un polynôme de degré trois. Maple n'a pas de méthode générale pour traiter ces équations :

```
isolve(y^2=x^3-2);
```

Dans de tels cas où le système échoue, il est souvent fructueux d'étudier l'équation modulo un entier $p$ avec la commande `msolve` :

```
msolve(y^2=x^3-2,8);
```

$$\{y=1,x=3\},\{y=3,x=3\},\{y=5,x=3\},\{y=7,x=3\}$$

On en déduit que $y$ doit être impair et $x$ congru à 3 modulo 8. La commande `isolve` fournit alors des solutions particulières :

```
seq(isolve({y^2=x^3-2,x=8*k+3}),k=0..10);
```

$$\{y=-5,x=3\},\{y=5,x=3\}$$

*Les nombres tricolores.* Les nombres tricolores sont les entiers dont le carré s'écrit uniquement avec des chiffres qui sont eux-mêmes des carrés, c'est-à-dire 1, 4 et 9. Par exemple 107 est tricolore car son carré vaut 11 449. Soit $n$ un entier tricolore, et $n=10q+r$ sa division euclidienne par 10. Alors $n^2 \bmod 10 = r^2 \bmod 10$, donc $r^2 \bmod 10 \in \{1,4,9\}$ :

```
seq(msolve(r^2=d,10),d={1,4,9});
```

$$\{r=1\},\{r=9\},\{r=2\},\{r=8\},\{r=3\},\{r=7\}$$

d'où le programme suivant qui calcule les $N$ premiers nombres tricolores :

```
tricol:=proc(N)
local k, res, q, r, n, i;
   k:=0;
   for q from 0 by 10 while k<N do
         for r in [1,2,3,7,8,9] do
            if is149((q+r)^2) then k:=k+1; res[k]:=q+r fi od od;
   [seq(res[i],i=1..N)]
end:
```

La fonction `is149` détermine si un entier donné s'écrit uniquement avec les chiffres 1, 4 et 9 :

```
is149:=proc(n:posint) local r;
```

```
        r:=n;
        while r<>0 do
            if not member(irem(r,10,'r'),{1,4,9}) then
                RETURN(false) fi od;
        true
    end:
```

Voici les dix premiers nombres tricolores :

```
tricol(10);
```

$$[\,1, 2, 3, 7, 12, 21, 38, 107, 212, 31488\,]$$

```
seq(x^2,x=");
```

$$1, 4, 9, 49, 144, 441, 1444, 11449, 44944, 991494144$$

### 1.5. Exercices.

1. On cherche à calculer le nombre de zéros finaux dans l'écriture de $n!$ en base 10. Écrire une procédure (prenant $n$ en argument) qui fait le calcul (en évitant de calculer $n!$). La tester sur diverses valeurs de $(10^i)!$.
2. Le calcul de $a^n$ peut être rendu encore plus rapide que par la méthode d'exponentiation binaire décrite en §1.1.1 si les divisions sont permises et rapides. Par exemple pour calculer $a^{31}$, la méthode d'exponentiation binaire effectue $a(a(a(aa^2)^2)^2)^2$, soit 4 produits et 4 carrés. Mais $a^{31} = a^{32}/a$, ce qui conduit au calcul $((((a^2)^2)^2)^2)^2/a$, soit 5 carrés et une division et un gain de 2 opérations. Écrire une procédure `expodiv` utilisant la division et comparer son efficacité avec `expo`.
3. Pour certaines valeurs de l'exposant $n$, le calcul de $a^n$ par la méthode d'exponentiation binaire n'est pas optimal. Par exemple, $a^{23} = s((st)^2)^2$, où $s = at$ et $t = a^2$. Le calcul de $a^{23}$ nécessite donc 6 multiplications, au lieu de 7 par la méthode d'exponentiation binaire. Écrire un programme prenant en entrée $n$ et renvoyant le nombre de multiplications minimal pour calculer $a^n$. Vérifier que l'on obtient 5 multiplications pour $n = 15$, 6 pour $n = 23$ et 8 pour $n = 63$. [Indication : commencer par généraliser le problème au calcul simultané d'un ensemble $\{a^{n_1}, \ldots, a^{n_k}\}$ de puissances, puis décomposer en fonction de la plus grande puissance.]
4. Trouver la signification du message codé

$$[\,96639780379700812966, 64982459586574831935\,]$$

fabriqué avec le produit $n = 289589985965965651459$ et l'exposant $e = 324803928042397$. Le code utilisé est RSA avec la transformation texte-entier de la page 98.

5. Pour calculer une fonction élémentaire $(\sin, \cos, \exp, \log)$ en précision fixe, les calculettes de poche utilisent une fonction approchée dont l'écart maximal avec la fonction sur l'intervalle donné est assez petit. Écrire une procédure calculant le sinus avec 10 décimales exactes. Il faut commencer par utiliser périodicité et symétrie pour se ramener au problème dans

$[0, \pi/2]$. Ensuite, il faut déterminer un approximant de PADÉ diagonal suffisamment bon sur l'intervalle. Enfin, optimiser le calcul par `convert[horner]`.

6. Spirale d'ULAM. Cette fameuse spirale est une représentation visuelle des nombres premiers. Pour la tracer, il faut se placer sur une feuille quadrillée. On part d'un carré, qui va être le centre de la spirale, et qui correspond à l'entier 1. Puis on se déplace vers l'un des quatre carrés adjacents, qui va représenter l'entier 2, et on continue ainsi en "tournant" autour des carrés déjà visités, en coloriant chaque carré correspondant à un nombre premier. Écrire un programme Maple traçant la partie de la spirale correspondant aux entiers inférieurs à $(2n+1)^2$. Il pourra s'avérer avantageux d'utiliser le crible d'ÉRATOSTHÈNE. La figure 1 montre le résultat obtenu avec $n = 128$.

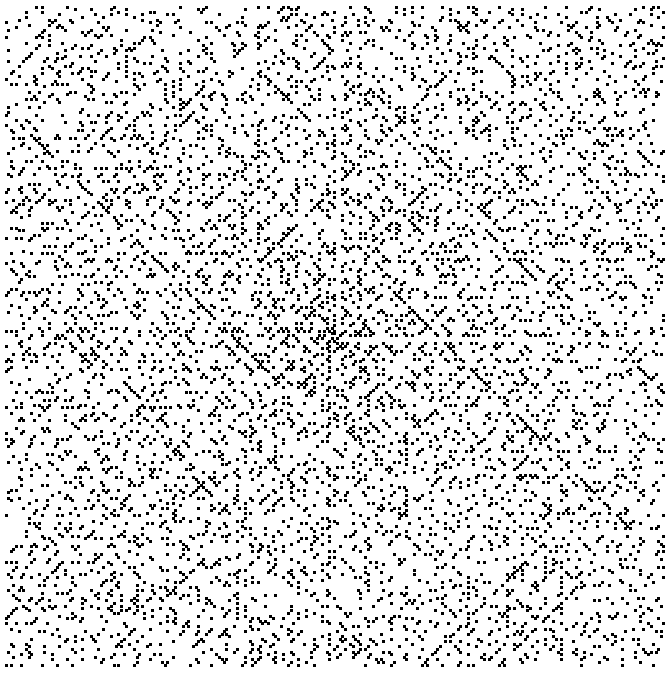


FIGURE 1 `plot(ulam(128),style=POINT,axes=NONE)`.

7. Les bovins d'Archimède Ce problème fut posé par Archimède à Ératosthène vers 200 av. J.-C. Il se réduit à la résolution du système

$$W = (\frac{1}{2} + \frac{1}{3})X + Z, \qquad X = (\frac{1}{4} + \frac{1}{5})Y + Z, \qquad Y = (\frac{1}{6} + \frac{1}{7})W + Z,$$
$$w = (\frac{1}{3} + \frac{1}{4})(X + x), \qquad x = (\frac{1}{4} + \frac{1}{5})(Y + y), \qquad y = (\frac{1}{5} + \frac{1}{6})(Z + z),$$
$$z = (\frac{1}{6} + \frac{1}{7})(W + w), \qquad m^2 = W + X, \qquad Y + Z = n(n+1)/2,$$

où $n, m$ sont des entiers, $W, X, Y, Z$ sont respectivement le nombre de taureaux blancs, noirs, tachetés, jaunes, et $w, x, y, z$ le nombre de vaches de ces couleurs.

À partir des sept équations linéaires, montrer qu'il existe un entier $k$ tel que $W = a_1 k, \ldots, z = a_8 k$ où $a_1, \ldots, a_8$ sont des entiers (de sept ou huit chiffres). En substituant dans l'équation $W + X = m^2$, montrer que $k$ est de la forme $4456749p^2$ ; puis en remplaçant dans $Y + Z = n(n+1)/2$, obtenir l'équation de Pell $N^2 = 4729494P + 1$ où $N = 2n + 1$ et $P = 9314p$, et la résoudre.
8. Pour rendre la recherche des nombres tricolores plus rapide, considérer les deux derniers chiffres des nombres. Vérifier que le programme obtenu est environ deux fois plus rapide que le premier.
9. Trouver les seize premiers nombres tricolores.

## 2. Combinatoire

La combinatoire s'intéresse à l'étude d'objets de taille finie. Ces objets sont généralement définis par un certain nombre de règles de construction. L'étude du nombre d'objets de taille donnée, ou de la valeur moyenne d'un paramètre de ces objets peut être grandement assistée par le calcul formel dans plusieurs directions. D'abord l'arithmétique en précision arbitraire facilite l'expérimentation. Ensuite le système connaît quelques objets de base de la combinatoire (permutations, partitions, compositions,...) dans son *package* `combinat`, que nous n'aborderons pas ici. Enfin, les capacités des systèmes à manipuler formellement des fonctions et séries peuvent être mises à contribution par le biais de *séries génératrices*. C'est sur cette approche que nous insisterons davantage.

**2.1. Approche empirique.** L'exemple des *dénumérants* permet de comparer l'approche empirique et l'utilisation de séries génératrices. Le problème est de compter le nombre $d_n$ de façons de totaliser $n$ francs avec des pièces de 1, 2, 5, 10 et 20 francs.

Le nombre $n_{20}$ de pièces de 20 francs peut varier de 0 à la partie entière de $n/20$, notée $\lfloor n/20 \rfloor$. Une fois ce nombre fixé, il reste à réaliser $p = n - 20n_{20}$ en pièces de 1, 2, 5 et 10 francs. Le problème est ainsi réduit à un problème plus simple. Ce procédé se réitère jusqu'à ce qu'il ne reste plus que des pièces de 1 franc. Le programme s'écrit donc

```
denumerants:=proc (n:nonnegint)
```

```
local n5, n10, n20, nb, p, q;
   nb:=0;
   for n20 from 0 to iquo(n,20) do
         p:=n−20*n20;
         for n10 from 0 to iquo(p,10) do
            q:=p−10*n10;
            for n5 from 0 to iquo(q,5) do
                  nb:=nb+iquo(q−5*n5,2)+1 od od od;
   nb
end:
```

Avec ce programme, voici le dénumérant de 1 000 et le temps de calcul nécessaire :

```
st:=time(): denumerants(1000),time()-st;
```

$$22457476, 18.850$$

**2.2. Structures décomposables.** Les techniques présentées dans cette section permettent de calculer le dénumérant $d_n$ comme coefficient de $z^n$ dans le développement de TAYLOR à l'origine de la fraction rationnelle

$$f = \frac{1}{1-z} \cdot \frac{1}{1-z^2} \cdot \frac{1}{1-z^5} \cdot \frac{1}{1-z^{10}} \cdot \frac{1}{1-z^{20}}. \tag{3}$$

La procédure devient

```
denumerants2:=proc (n:nonnegint)
local f,z;
   f:=1/(1−z)/(1−z^2)/(1−z^5)/(1−z^10)/(1−z^20);
   coeff(series(f,z,n+1),z,n)
end:
```

Et le temps de calcul est nettement plus faible :

```
st:=time(): denumerants2(1000),time()-st;
```

$$22457476, 5.116$$

De plus, alors que `denumerants` ne calcule qu'un seul $d_n$ à la fois, la procédure `denumerants2`, en faisant le développement limité de $f$, les calcule tous jusqu'à l'ordre $n$.

Pour comprendre pourquoi le résultat de `denumerants2` vaut $d_n$, il suffit de détailler le développement de chacun des termes de $f$ :

$$(1+z+z^2+\cdots)(1+z^2+z^4+\cdots)(1+z^5+\cdots)(1+z^{10}+\cdots)(1+z^{20}+\cdots).$$

Lors du développement du produit, la puissance $n_1$ de $z$, la puissance $n_2$ de $z^2$,..., la puissance $n_{20}$ de $z^{20}$ donnent $z$ élevé à la puissance $n_1 + 2n_2 + 5n_5 + 10n_{10} + 20n_{20}$. Toutes les $d_n$ façons d'obtenir l'exposant $n$ vont ainsi contribuer d'une unité au coefficient de $z^n$.

La raison pour laquelle ce programme est plus rapide que le précédent est double. D'une part la commande `series` fait partie du noyau de Maple, et est donc d'exécution très rapide[†]. D'autre part, les produits intermédiaires

[†]Une liste des procédures du noyau est donnée au chapitre II, §3.3.

fournissent des dénumérants partiels qui ne sont ainsi calculés qu'une fois. Le calcul effectué par `series` est ainsi le même que celui qu'effectue le programme suivant, légèrement plus rapide que `denumerants`, auquel on donne la liste de pièces en second argument :

```
denumerants1:=proc(n:nonnegint,l:list(posint))
option remember;
local res, p, k, i, newl;
   if l=[1] then RETURN(1) fi;
   k:=l[nops(l)];
   newl:=subsop(nops(l)=NULL,l);
   p:=iquo(n,k);
   for i from 0 to p do res[i]:=denumerants1(n−k*i,newl) od;
   convert([seq(res[i],i=0..p)],`+`)
end:
```

2.2.1. *Structures décomposables*

L'exemple des dénumérants peut être reformulé dans un cadre plus général, celui des classes de structures décomposables. Une structure est dite décomposable si elle est formée à partir d'atomes par un nombre fini de constructions combinatoires. Les constructions combinatoires les plus couramment utilisées sont l'union, le produit cartésien et la construction "séquence", qui forme des suites d'objets. L'exemple des dénumérants se décompose en le produit cartésien d'une séquence (éventuellement vide) de pièces de 1 franc, d'une séquence de pièces de 2 francs,..., et d'une séquence de pièces de 20 francs, ce que l'on note

$$D = \mathrm{Seq}(F_1) \times \mathrm{Seq}(F_2) \times \mathrm{Seq}(F_5) \times \mathrm{Seq}(F_{10}) \times \mathrm{Seq}(F_{20}) \tag{4}$$

où $F_k$ est l'atome "pièce de $k$ francs".

L'intérêt d'une telle formulation est qu'elle se traduit directement en termes de *séries génératrices*. La série génératrice de la suite $u_n$ est par définition

$$U(z) = \sum_{n=0}^{\infty} u_n z^n.$$

Si $A$ est une classe de structures décomposables, $a_n$ représente le nombre de structures de taille $n$ dans cette classe, et $A(z)$ la fonction génératrice de cette suite. Les règles données au tableau 1 permettent alors de calculer facilement les séries génératrices.

Ainsi l'équation (4) se traduit en

$$D(z) = \frac{1}{1-F_1(z)} \cdot \frac{1}{1-F_2(z)} \cdot \frac{1}{1-F_5(z)} \cdot \frac{1}{1-F_{10}(z)} \cdot \frac{1}{1-F_{20}(z)}.$$

La taille d'une pièce étant prise égale à sa valeur, $F_k(z) = z^k$, ce qui donne bien la forme indiquée en (3). Le coefficient de $z^n$ dans la série $D(z)$ est le nombre de structures de taille $n$, c'est-à-dire ici le nombre de combinaisons de valeur $n$

TABLE 1 Correspondance entre constructions combinatoires et séries génératrices.

| Structure | Série génératrice |
|---|---|
| $A \cup B$ | $A(z) + B(z)$ |
| $A \times B$ | $A(z) \cdot B(z)$ |
| $\mathrm{Seq}(A)$ | $1/(1 - A(z))$ |

francs. Si la taille d'une pièce est prise égale à 1, c'est-à-dire $F_k(z) = z$, le $n^{\mathrm{e}}$ coefficient de $D(z)$ devient le nombre de façons de faire $n$ pièces avec des pièces de cinq types différents.

Les règles du tableau 1 p. 109 peuvent être appliquées à des structures définies récursivement. Par exemple, les arbres binaires se décomposent en

$$B = \text{Feuille} \ \cup \ (B \times B).$$

Autrement dit, un arbre binaire est soit une feuille, soit un nœud dont partent deux arbres binaires. En prenant comme notion de taille le nombre de feuilles, l'application des règles du tableau 1 p. 109 donne pour les séries génératrices

$$B(z) = z + B^2(z).$$

Cette équation se résout :

```
solve(B=z+B^2,B);
```

$$\frac{1}{2} - \frac{1}{2}\sqrt{1-4\,z}, \quad \frac{1}{2} + \frac{1}{2}\sqrt{1-4\,z}$$

```
map(series,["],z);
```

$$[z + z^2 + 2z^3 + 5z^4 + 14z^5 + \mathrm{O}(z^6), 1 - z - z^2 - 2z^3 - 5z^4 - 14z^5 + \mathrm{O}(z^6)]$$

Seule la première solution n'a que des coefficients positifs et est donc la série cherchée. La commande `series` permet alors de calculer rapidement de nombreux coefficients.

Nous avons présenté ici une version très simplifiée de la théorie des structures décomposables. D'autres constructions sont utilisées, en particulier la formation d'ensembles et de cycles, ce qui produit des fonctions exp et log pour les séries génératrices. On peut aussi créer des séries génératrices à plusieurs variables, où chaque variable compte un paramètre donné. Ainsi le remplacement de $F_k(z)$ par $uz^k$ dans $D(z)$ donne une série génératrice dans laquelle le coefficient de $u^p z^n$ est le nombre de façons de faire $n$ francs avec $p$ pièces de valeur 1, 2, 5, 10 ou 20 francs.

Le calcul formel est utile dans la phase de résolution des équations de séries génératrices, mais surtout dans la phase de calcul des coefficients. Il permet aussi d'étudier le comportement asymptotique des coefficients ; nous y reviendrons au chapitre VIII.

2.2.2. *Exemple : les décompositions de Lagrange*

Étant donné un entier $n$, il s'agit de trouver le nombre $q_n$ de quadruplets $(a, b, c, d)$ d'entiers positifs ou nuls tels que $a^2 + b^2 + c^2 + d^2 = n$. La méthode des séries génératrices s'applique : la classe $Q$ des quadruplets est le produit cartésien $C \times C \times C \times C$, où l'ensemble $C$ des carrés est la réunion des $C_k$ et $C_k$ désigne le carré de l'entier $k$. L'application des règles du tableau 1 p. 109 donne pour les séries génératrices $Q(z) = C^4(z)$ et $C(z) = \sum C_k(z)$. La taille d'un entier est ici sa valeur, soit $C_k(z) = z^{k^2}$. Le nombre $q_n$ de décompositions de LAGRANGE de $n$ est donc le coefficient de $z^n$ dans

$$\left(\sum_{k=0}^{\infty} z^{k^2}\right)^4 .$$

En pratique, pour calculer un $q_n$ donné, par exemple pour $n = 1\ 000$, il faut tronquer explicitement $C(z)$ à l'indice $\sqrt{n}$.

```
n:=1000: C:=convert([seq(z^(k^2),k=0..isqrt(n))],`+`):
coeff(series(C^4,z,n+1),z,n);
```

276

2.2.3. *Le problème des reines*

L'exemple précédent n'utilisait qu'une variable. Voici un exemple plus compliqué où le nombre de variables dépend de la taille du problème.

Combien y a-t-il de façons de placer $n$ reines sur un échiquier $n \times n$, de façon qu'elles soient toutes sur des lignes, colonnes et diagonales différentes ? Pour $n = 8$, il y a 92 solutions, dont une est représentée figure 2.

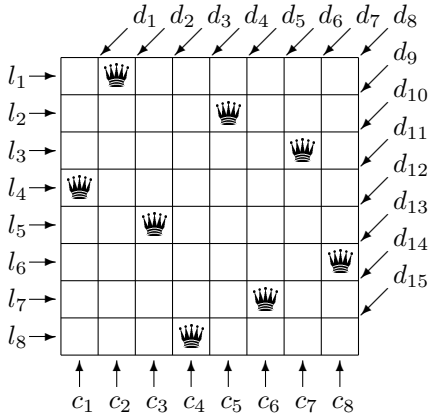



FIGURE 2 L'une des 92 solutions au problème des 8 reines.

À cause du nombre de contraintes, la solution à ce problème n'est pas immédiate en termes de séries génératrices. En revanche, la série génératrice de toutes les dispositions possibles de reines sur l'échiquier s'obtient facilement. Il ne reste plus alors qu'à lui enlever les termes indésirables.

Comme l'échiquier comporte $n^2$ cases, une disposition de reines est le produit cartésien de $n^2$ états individuels. La case $(i,j)$ est soit vide, soit occupée par une reine. Cela donne pour la série génératrice des dispositions

$$P = \prod_{1\leq i,j\leq n} (1 + D_{i,j}),$$

où le terme 1 représente une case vide, et $D_{i,j}$ code l'occupation de la case $(i,j)$. Pour pouvoir éliminer les mauvaises dispositions, on remplace $D_{i,j}$ par $l_i c_j d_{i+j-1} e_{i-j+n}$ : la variable $l_i$ code l'occupation de la ligne $i$, $c_j$ celle de la colonne $j$, $d_{i+j-1}$ celle de la diagonale montante et $e_{i-j+n}$ celle de la diagonale descendante passant par la case $(i,j)$. Chaque terme du développement du polynôme $P$ devient alors un monôme formé de variables $l_i, c_j, d_k, e_l$. La configuration de la figure 2 correspond ainsi au monôme

$$l_4c_1d_4e_{11}\ l_1c_2d_2e_7\ l_5c_3d_7e_{10}\ l_8c_4d_{11}e_{12}\ l_2c_5d_6e_5\ l_7c_6d_{12}e_9\ l_3c_7d_9e_4\ l_6c_8d_{13}e_6.$$

Cette correspondance n'est pas bijective : par exemple pour $n = 4$, le monôme $l_1l_2l_3l_4c_1c_2c_3c_4d_2d_3d_5d_6e_2e_3e_5e_6$ correspond à deux configurations différentes. Les "bons" monômes sont ceux sans carré (sinon une ligne, colonne ou diagonale contient plusieurs reines) et contenant chacune des variables $l_1, \ldots, l_n$, $c_1, \ldots, c_n$ (ce qui fixe le nombre de reines à $n$). Le nombre de solutions est donc la somme des coefficients des monômes sans carré du coefficient de $l_1 \cdots l_n, c_1 \cdots c_n$ dans

$$P_n = \prod_{i=1}^{n}\prod_{j=1}^{n} (1 + l_i c_j d_{i+j-1} e_{i-j+n}). \tag{5}$$

Le développement brutal de ce produit mène à une explosion combinatoire : $P_n$ comporte $2^{n^2}$ termes, soit déjà 33 554 432 pour $n = 5$ !

L'extraction directe du coefficient de $l_i$ dans (5) limite cette explosion : le calcul est réduit à la recherche du coefficient de $c_1 \cdots c_n$ dans

$$P'_n = \prod_{i=1}^{n}\sum_{j=1}^{n} c_j d_{i+j-1} e_{i-j+n}.$$

Le nombre de termes est passé de $2^{n^2}$ à $n^n$. Symboliquement, on réduit encore cette explosion en développant successivement par rapport à chacune des variables $c_i$ et en limitant la suite du calcul au coefficient de $c_i$. C'est ce que fait la procédure suivante :

```
reines := proc(n)
local i, j, P, c, d, e, L;
   P:=convert([seq(convert(
         [seq(c[j]*d[i+j-1]*e[i-j+n],j=1..n)],`+`),i=1..n)],§`*`);
   for i to n do P:=coeff(series(P,c[i],2),c[i],1) od;
   L:=[seq(e[i]^2=0,i=1..2*n-1),seq(d[i]^2=0,i=1..2*n-1)];
   P:=expand(subs(L,P));
   P:=P-select(hastype,P,§`^`);
```

```
        subs([seq(e[i]=1,i=1..2*n−1),seq(d[j]=1,j=1..2*n−1)],P)
    end:
seq(reines(n),n=4..8);
```

$$2, 10, 4, 40, 92$$

La valeur 92 fut trouvée il y a près d'un siècle et demi. Le programme peut être encore un peu amélioré en éliminant les carrés au fur et à mesure du calcul, mais cela ne permet guère d'aller plus loin que $n = 8$.

Cet exemple montre à la fois la puissance du calcul formel — le problème des reines se résout par un programme Maple de six lignes — et une de ses limitations — la taille des expressions rend rapidement les calculs infaisables.

### 2.2.4. *Récurrences linéaires*

Les dénumérants peuvent se calculer de manière encore plus rapide que par le développement de leur série génératrice. En effet, les coefficients de fractions rationnelles satisfont des récurrences linéaires à coefficients constants. Connaissant la série génératrice $f(z) = p(z)/q(z)$, où $p$ et $q$ sont deux polynômes, la récurrence s'écrit directement à l'aide des coefficients de $p$ et $q$ : il suffit d'écrire la nullité du coefficient de $z^n$ dans $qf - p$, soit :

$$q_0 f_n + q_1 f_{n-1} + \cdots q_k f_{n-k} = 0, \qquad n > \max(\deg p, \deg q - 1),$$

en notant

$$q(z) = q_0 + q_1 z + \cdots + q_k z^k.$$

Il s'ensuit une procédure très générale pour calculer le développement de fractions rationnelles[†] :

```
series_ratpoly:=proc(f,z:name,n:posint)
local g, p, q, dp, i, u, k, j, uj;
  if not type(f,ratpoly(anything,z)) then
        ERROR(§‘invalid argument‘,f) fi;
  g:=normal(f); p:=collect(numer(g),z); q:=collect(denom(g),z);
  if subs(z=0,q)=0 then ERROR(§‘invalid argument‘,f) fi;
  # récurrence :
  uj:=convert([seq(−coeff(q,z,i)/coeff(q,z,0)*u[j−i],
        i=1..degree(q,z))],‘+‘);
  # premiers termes :
  dp:=max(degree(p,z),degree(q,z)-1);
  g:=series(f,z,dp+1);
  for i from 0 to dp do u[i]:=coeff(g,z,i) od;
  # fin du développement :
  for j from dp+1 to n do u[j]:=eval(uj) od;
  # et on construit le résultat :
  series(convert([seq(u[j]*zˆj,j=0..n),O(zˆ(n+1))],‘+‘),z,n+1)
```

[†]Noter l'utilisation subtile de `eval` pour n'évaluer qu'une fois les coefficients de la récurrence sans créer de procédure.

**end**:

Lorsqu'elle est appliquée à la série génératrice des dénumérants, cette procédure donne un temps de calcul encore meilleur que le précédent :

```
f:=1/(1-z)/(1-z^2)/(1-z^5)/(1-z^10)/(1-z^20):
st:=time(): coeff(series_ratpoly(f,z,1000),z,1000),time()-st;
```

$$22457476, \quad 1.750$$

Cela signifie que la procédure du noyau Maple qui calcule des développements en série n'a pas de cas particulier pour les fractions rationnelles, dont elle calcule le développement en faisant une division de développements, opération dont la complexité est plus élevée.

Cette méthode est la meilleure pour obtenir toutes les valeurs de la suite jusqu'à la $n^{\text{e}}$. Pour obtenir uniquement la $n^{\text{e}}$ valeur, il existe une méthode plus rapide qui est décrite au chapitre VII. La correspondance entre les fractions rationnelles et les récurrences linéaires à coefficients constants y est également exploitée pour l'étude asymptotique des suites solutions de ces récurrences.

La transformation de série génératrice en récurrence linéaire présentée ici se généralise. Les coefficients de fonctions algébriques, et plus généralement de solutions d'équations différentielles linéaires à coefficients polynomiaux, vérifient aussi des récurrences linéaires à coefficients qui sont polynomiaux en $n$ alors qu'ils sont constants dans le cas des fractions rationnelles[†]. Les propriétés analytiques des solutions de ces équations différentielles peuvent être exploitées pour l'étude — notamment asymptotique — des suites.

En conclusion, dans le cas de structures décomposables, la série génératrice, qu'elle soit donnée sous forme explicite ou implicite, permet de calculer le nombre $f_n$ de solutions pour $n$ donné. Dans certains cas le comportement asymptotique de $f_n$ lorsque $n$ tend vers l'infini peut également s'en déduire (voir VII§1.1.3 et VIII§3).

**2.3. Le problème des obèses.** Nous détaillons la résolution d'un problème qui ne repose pas sur une structure décomposable, mais où, comme dans le problème des reines, le coût de l'étude est réduit grâce à un codage par des polynômes.

Des hommes très corpulents viennent s'asseoir tour à tour sur une rangée de $n$ chaises initialement toutes libres. Chacun a besoin de deux places contiguës, et choisit ces places de façon aléatoire et uniforme parmi les paires de places libres adjacentes. Combien de chaises libres reste-t-il lorsqu'aucun homme ne peut plus s'asseoir ?

Ce problème n'est pas décomposable. En effet, l'occupation d'une chaise n'est pas indépendante de celle des chaises adjacentes, et par conséquent il n'est pas possible de décomposer une configuration. Cependant, comme pour le problème des reines, il s'agit d'un problème d'occupation avec contraintes qui se code par des polynômes.

---

[†]Ces transformations ne sont pas encore implantées de façon standard dans les systèmes de calcul formel actuels. On peut les trouver dans le *package* `gfun` de Maple.

L'ensemble des configurations possibles est représenté par un arbre. La racine représente la configuration initiale où toutes les chaises sont vides (voir fig. 3). Chaque chemin depuis la racine jusqu'à un nœud a une certaine probabilité d'occurrence, qui est indiquée à côté de la configuration. La somme des probabilités à chaque niveau de l'arbre (c'est-à-dire dans chaque colonne de la figure 3) vaut 1. Les feuilles de l'arbre sont les configurations terminales, telles qu'il ne reste pas deux places adjacentes. Cet arbre se traduit ensuite

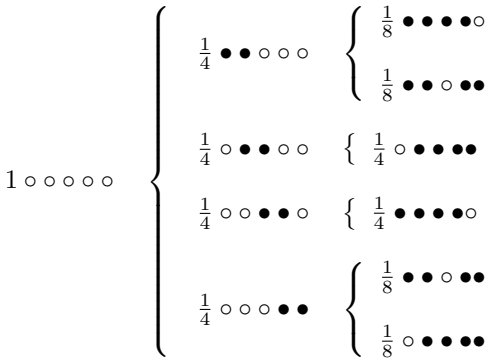



FIGURE 3 L'arbre des configurations pour $n = 5$.

en un polynôme : chaque chaise occupée est associée à une variable $x_k$ ; une configuration devient un monôme, par exemple $x_1x_2x_4x_5$ indique que les chaises 1 et 2 sont occupées ainsi que les chaises 4 et 5. La somme des monômes correspondants aux nœuds du niveau $j$ de l'arbre multipliés par les probabilités associées donne un polynôme codant toutes les configurations possibles après que $j$ hommes se sont assis. Au niveau 1 et pour $n = 5$ ce polynôme vaut

$$\frac{1}{4}x_1x_2 + \frac{1}{4}x_2x_3 + \frac{1}{4}x_3x_4 + \frac{1}{4}x_4x_5$$

et au niveau 2 :

$$\frac{3}{8}x_1x_2x_3x_4 + \frac{1}{4}x_1x_2x_4x_5 + \frac{3}{8}x_2x_3x_4x_5. \tag{6}$$

L'utilisation de polynômes permet ainsi de regrouper les configurations identiques obtenues par des chemins différents dans l'arbre, tout en sommant leurs probabilités respectives. Les six feuilles de l'arbre de la figure 3 sont représentées par 3 monômes dans (6), qui correspondent aux trois configurations terminales possibles. Pour $n = 10$, l'arbre comporte 366 feuilles pour seulement 12 configurations finales.

En termes informatiques, la méthode présentée ici revient à faire un parcours en largeur de l'arbre, avec factorisation des configurations semblables ; la même technique fut déjà employée pour le problème des reines.

L'implantation en Maple s'effectue en deux temps. Une première procédure calcule, à partir d'une configuration donnée sous la forme d'un monôme $m$,

les nouvelles configurations possibles et leurs probabilités. Ces configurations sont obtenues en recherchant les paires de variables $(x_{i-1}, x_i)$ absentes du monôme. S'il ne reste pas de places adjacentes, le monôme correspond à une feuille et contribue à l'espérance cherchée : son coefficient est la probabilité de la feuille et le nombre de places vides est $n$ moins son degré. Les feuilles sont quant à elles codées comme des monômes en la variable $z$.

```
iteration := proc(m,x,n,z) # x est la liste des x[i]
local i,j,s;
   if has(m,z) then RETURN(m) fi;
   j:=0; s:=0;
   for i from 2 to n do if not has(m,{x[i−1],x[i]}) then
         j:=j+1; s:=s+m*x[i−1]*x[i] fi od;
   if j<>0 then s/j
   else z*(n−degree(m,x))*subs([seq(x[i]=1,i=1..n)],m) fi
end:
```

Le programme principal prend en entrée le nombre $n$ de chaises, et calcule le nombre moyen de chaises restant à la fin. Il suffit de partir du monôme 1 représentant la configuration initiale et d'appliquer la procédure `iteration` à chaque monôme. Le programme s'arrête lorsque tout l'arbre a été parcouru.

```
fatmen:=proc(n)
local l, x, i, vars, z;
   l:=1; vars:=[seq(x[i],i=1..n)];
   while subs(z=0,l)<>0 do l:=map(iteration,l,vars,n,z) od;
   subs(z=1,l)
end:
```

```
seq(fatmen(n),n=3..10);
```

$$1, \frac{2}{3}, 1, \frac{16}{15}, \frac{11}{9}, \frac{142}{105}, \frac{67}{45}, \frac{4604}{2835}$$

Pour une rangée de 10 chaises, le nombre moyen de places vides est donc à peu près 1,62 lorsque plus aucun homme ne peut s'asseoir. On peut montrer (mais cela dépasse le cadre de cet ouvrage) que la série génératrice associée aux nombres `fatmen(n)` est $x\exp(-2x)/(1-x)^2$. On en déduit que lorsque $n$ tend vers l'infini, la proportion de places libres tend vers $\exp(-2) \simeq 0{,}135$.

### 2.4. Exercices.

1. Modifier le programme `denumerant` pour qu'il affiche les différentes décompositions de $n$.
2. Utiliser la méthode du §2.1 pour déterminer le nombre de façons de totaliser un kilogramme avec des poids de 50, 100, 200 et 500 grammes. Que devient la réponse si le nombre de poids de chaque sorte est limité, par exemple à 3 ?
3. Les entiers *bicarrés* sont ceux qui sont de la forme $a^2 + b^2$ avec $a$ et $b$ entiers positifs ou nuls. En utilisant la méthode décrite en §2.2.2, écrire

un programme donnant le nombre $N(x)$ d'entiers bicarrés inférieurs à $x$. Vérifier empiriquement que $N(x)$ se comporte asymptotiquement comme $Cx/\sqrt{\log x}$ où $C \simeq 0{,}764$.

4. Boules dans des urnes. Utiliser la théorie des structures décomposables pour calculer le nombre de façons de mettre $n$ boules :
   (1) dans 5 urnes pour $n = 3, 10, 100$ ;
   (2) dans 10 urnes pour $n = 3, 10, 100$ ;
   (3) dans $k$ urnes pour $n = 3, 10, 100$ et $k = 5, 10, 15, 20$ [Utiliser une fonction génératrice à deux variables.] ;
   (4) dans $k$ urnes lorsque chacune contient au plus 10 boules, pour $n = 3, 10, 100$ et $k = 5, 10, 15, 20$ ;
   (5) dans $k$ urnes lorsque chacune contient au plus $r$ boules, pour les mêmes valeurs de $n$ et $k$, et pour $r = 1, 2, 5, 10, 20$. [Utiliser une fonction génératrice à trois variables.]
5. Composition d'entiers. Utiliser la théorie des structures décomposables pour calculer le nombre de façons d'écrire l'entier $n$ comme :
   (1) somme de deux entiers (positifs), pour $n \leq 1\ 000$ (on tient compte de l'ordre des sommants, ainsi 1+2 et 2+1 sont deux sommes différentes) ;
   (2) somme de $k$ entiers, pour $n \leq 1\ 000$ et $k = 2, 3, 5, 10$ [Utiliser une fonction génératrice à deux variables.] ;
   (3) somme de $k$ entiers impairs, pour les mêmes valeurs de $n$ et $k$ ;
   (4) somme de $k$ entiers congrus à 1 modulo $p$, pour les mêmes valeurs de $n$ et $k$, et pour $p = 2, 3, 5, 10$ [Utiliser une fonction génératrice à trois variables.]
6. Suites de pile ou face. Utiliser la théorie des structures décomposables pour calculer le nombre de suites de pile ou face de longueur $n$ :
   (1) sans contrainte, pour $n \leq 100$ ;
   (2) lorsque pile n'arrive jamais deux fois consécutives, pour $n \leq 100$ ;
   (3) lorsque pile n'arrive jamais $k$ fois consécutives, pour $n \leq 100$ et $k = 2, 5, 10$.
7. Arbres. Reprendre le calcul de la série génératrice des arbres binaires pour calculer le nombre d'arbres binaires de taille $n$, pour $n \leq 100$ :
   (1) lorsque la taille est le nombre de feuilles ;
   (2) lorsque la taille est le nombre de nœuds ;
   (3) lorsque la taille est la somme du nombre de feuilles et du nombre de nœuds.

   Mêmes questions pour des arbres ternaires (les nœuds ont trois descendants), les arbres unaires-binaires (les nœuds ont un ou deux descendants), et les arbres à arité quelconque (les nœuds ont un nombre quelconque de descendants).

CHAPTER V

# Calcul matriciel

Le calcul matriciel au sens large est un domaine où un système de calcul formel peut apporter beaucoup. En effet, au delà de matrices $3 \times 3$, il devient difficile de réaliser des calculs à la main. La complexité de ces calculs n'est pourtant pas très élevée, ce qui permet aux systèmes de calcul formel de résoudre des problèmes de taille assez importante.

## 1. Matrices et vecteurs

Tous les systèmes de calcul formel savent manipuler des matrices et des vecteurs, seule la syntaxe est différente selon les systèmes.

En Maple, tout ce qui a trait à l'algèbre linéaire se trouve dans le *package* `linalg` ; pour alléger les notations, nous supposerons dans la suite du chapitre qu'il a été chargé, ce qui s'effectue par `with(linalg)`[†].

Les fonctions du *package* `linalg` s'appliquent principalement à des matrices dont les éléments sont des fractions rationnelles. Par ailleurs les matrices considérées doivent avoir une dimension bien définie : les systèmes de calcul formel ne savent pas travailler sur des matrices dont la dimension est un paramètre.

**1.1. Les objets vecteur et matrice en Maple.** Un vecteur est un tableau à une dimension dont les indices commencent à 1 et une matrice est un tableau à deux dimensions dont les indices commencent à 1.

La fonction `array` permet de créer des matrices et des vecteurs, mais il vaut mieux utiliser les deux fonctions `vector` et `matrix` dont la syntaxe est plus souple.

Exemple 1. Voici deux vecteurs, le premier défini par ses coordonnées, le second par une fonction permettant de les calculer :

```
vector([1,2]), vector(3,i->x*i);
```

$$\left[\begin{array}{cc} 1 & 2 \end{array}\right], \quad \left[\begin{array}{ccc} x & 2x & 3x \end{array}\right]$$

Exemple 2. Les matrices sont créées de la même façon :

```
matrix([[1,2],[3,4]]), matrix(2,3,(i,j)->x^(i+j-1));
```

[†]Deux fonctions de Maple sont alors "écrasées" et remplacées par celles du *package* : ce sont `norm` qui calcule la norme d'un polynôme et surtout `trace` qui affiche les étapes successives de l'exécution d'une fonction. Il est possible d'y accéder tout de même (voir II§3.3).

TABLE 1 Vecteurs et matrices.

| définition | types | représentation interne |
|---|---|---|
| `vector(n,...)` | `table`<br>`array`<br>`vector` | `array(1..n,...)` |
| `matrix(m,n,...)` | `table`<br>`array`<br>`matrix` | `array(1..m,1..n,...)` |

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \qquad \begin{bmatrix} x & x^2 & x^3 \\ x^2 & x^3 & x^4 \end{bmatrix}$$

Un vecteur en Maple est un tableau unidimensionnel. Il est considéré dans les calculs matriciels comme un vecteur colonne, c'est-à-dire comme une matrice à $n$ lignes et une colonne. Mais *ce n'est pas* une matrice (qui est un tableau bidimensionnel). Cela implique en particulier qu'il n'est pas du type `matrix`. En outre, l'affichage d'un vecteur se fait sous forme d'une ligne, ce qui peut entraîner des confusions. Enfin, l'application de la fonction `transpose` à un vecteur ne donne pas un vecteur ligne, qui n'existe pas en tant que tel en Maple. Un vecteur ligne ne peut être en effet qu'une matrice à une ligne et $n$ colonnes. Le transposé du vecteur `v` s'utilise sous la forme `transpose(v)` qui est reconnue dans les calculs matriciels. Le tableau 1 résume ces différences entre vecteurs et matrices.

EXEMPLE 3. Nous définissons un vecteur de dimension 2 et une matrice $2 \times 2$ :

```
v:=vector([1,2]): m:=matrix(2,2,[[1,2],[3,4]]):
multiply(m,v);
```

$$\begin{bmatrix} 5 & 11 \end{bmatrix}$$

```
transpose(v), multiply(v,transpose(v));
```

$$\text{transpose}(v), \quad \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$$

Maple réalise bien le produit de $m$ par $v$ et le résultat est un vecteur ; le terme `transpose(v)` est laissé tel quel, mais est reconnu comme un vecteur ligne dans $v^t v$.

Une autre difficulté que l'on rencontre lors de la manipulation de vecteurs et de matrices est liée à la règle d'évaluation de ces objets (voir II§3.1). En particulier, il faut parfois faire explicitement appel à `op` ou `eval` pour accéder à leur contenu.

EXEMPLE 4.

```
a:=matrix([[b,2],[3,4]]): a;
```

$$a$$

```
print(a);
```

$$\begin{bmatrix} b & 2 \\ 3 & 4 \end{bmatrix}$$

```
subs(b=1,a), subs(b=1,op(a));
```

$$a, \quad \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
transpose(a), transpose(op(a));
```

$$\begin{bmatrix} b & 3 \\ 2 & 4 \end{bmatrix}, \quad \begin{bmatrix} b & 3 \\ 2 & 4 \end{bmatrix}$$

**1.2. Manipulation des matrices.** Dans tous les systèmes de calcul formel, il existe une panoplie de fonctions pour réaliser les opérations matricielles de base. En Maple les principales fonctions de *manipulation* (c'est-à-dire qui n'effectuent pas de calcul) proposées par le *package* `linalg` sont résumées au tableau 2.

TABLE 2 Principales manipulations de matrices.

| Fonction | Commande |
|---|---|
| Transposition | `transpose` |
| Création d'une matrice bloc-diagonale | `diag` |
| Création d'une matrice bande | `band` |
| Création d'une matrice compagnon | `companion` |
| Extraction d'une sous-matrice | `submatrix` |
| Extraction de colonnes | `col` |
| Extraction de lignes | `row` |
| Adjonction de colonnes | `augment` |
| Adjonction de lignes | `stack` |
| Suppression de colonnes | `delcols` |
| Suppression de lignes | `delrows` |

Les fonctions `map` et `zip`, qui s'appliquent à d'autres objets que les matrices et les vecteurs peuvent également s'avérer utiles.

EXEMPLE 5. La commande `map` applique une fonction aux éléments d'un vecteur ou d'une matrice.

```
m:=matrix(2,2,(i,j)->x^(i+j));
```

$$m := \begin{bmatrix} x^2 & x^3 \\ x^3 & x^4 \end{bmatrix}$$

```
map(diff,m,x), diff(m,x);
```

$$\begin{bmatrix} 2x & 3x^2 \\ 3x^2 & 4x^3 \end{bmatrix}, \quad 0$$

Si `map` n'est pas utilisé, `m` a pour valeur son nom `m`, variable indépendante de $x$ dont la dérivée par rapport à `x` vaut 0. En revanche, `diff(eval(m),x)` produit un message d'erreur explicite.

EXEMPLE 6. La fonction `zip` applique une fonction binaire à deux vecteurs. Pour calculer un vecteur dont chaque élément est le pgcd des éléments de deux vecteurs, il suffit d'effectuer :

```
v1:=vector([x^2,-x^3+1,x^2-1]); v2:=vector([x,x-1,x^2+3*x+2]);
```

$$v1 := \left[\begin{array}{ccc} x^2 & -x^3+1 & x^2-1 \end{array}\right]$$

$$v2 := \left[\begin{array}{ccc} x & x-1 & x^2+3x+2 \end{array}\right]$$

```
zip(gcd,v1,v2), gcd(v1,v2);
```

$$\left[\begin{array}{ccc} x & x-1 & x+1 \end{array}\right], \qquad 1$$

Là encore la fonction `gcd` appliquée à `v1` et `v2` donne 1 car l'évaluation de `v1` et `v2` produit leur nom, à moins d'utiliser explicitement `eval` ou `op`.

**1.3. Calculs matriciels de base.** Les opérations de base sur les matrices (addition, produit, produit par un vecteur,...) sont fournies par le package *linalg* comme nous l'avons dit. Cependant leur notation fonctionnelle rend l'écriture des expressions matricielles assez malaisée. Par exemple, l'expression ${}^t x(Ax + 2y)$ peut s'écrire :

```
multiply(transpose(x),add(multiply(A,x),scalarmul(y,2))).
```

L'évaluateur `evalm`, qui ne fait pas partie du *package* `linalg` et est disponible en standard, autorise une syntaxe plus simple. L'expression précédente s'écrit alors :

```
evalm(transpose(x) &* (A &* x + 2 * y))
```

Le tableau 3 décrit les différentes opérations de base, aussi bien en utilisant `evalm` que le *package* `linalg`.

TABLE 3 Opérations sur des matrices.

| | `evalm` | *package* `linalg` |
|---|---|---|
| addition | `A + B` | `add(A,B)` |
| multiplication par un scalaire | `n * A` | `scalarmul(A,n)` |
| multiplication matricielle | `A &* B` | `multiply(A,B)` |
| inverse | `A^(-1)` | `inverse(A)` |
| puissance entière | `A^n` | `multiply(A,...,A)` |
| identité | `&*()` | `band([1],n)` |
| matrice diagonale constante | `k` | `band([k],n)` |

Il est à noter que `&*()` représente pour `evalm` l'élément neutre de la multiplication des matrices, donc la matrice identité. Il n'est pas nécessaire de définir sa taille. Il en est de même lors de `evalm(k + A)`, où l'expression `k` est interprétée comme $k$ fois la matrice identité.

**1.4. Exercices.**

1. Créer les trois objets suivants : une liste `l` ayant pour éléments 1, 2, 3, un vecteur `v` ayant les mêmes éléments et une matrice ligne `m` ayant aussi les mêmes éléments. Comparez l'affichage de ces différents objets.

2. Trouver une condition nécessaire et suffisante pour que deux matrices $2 \times 2$ symétriques commutent.
3. Créer une matrice $R$ de rotation autour de l'origine. Multiplier $R$ par elle-même, et vérifier que le résultat correspond à la rotation d'angle double.

## 2. Algèbre linéaire

Un grand nombre d'applications d'algèbre linéaire se ramènent à des calculs matriciels, en particulier à la résolution de systèmes d'équations linéaires, dont seule la taille empêche le traitement à la main.

**2.1. Résolution de systèmes linéaires.** Pour de gros systèmes, c'est-à-dire à l'heure actuelle ceux dont la taille dépasse 25 équations à 25 inconnues, il n'existe pas d'autre recours que les techniques du calcul numérique qui permettent de traiter des systèmes denses de taille $100 \times 100$ et des systèmes creux de taille $1000 \times 1000$. Les systèmes de calcul formel ont peu à proposer pour des problèmes de cette taille. Cependant les techniques numériques sont souvent sensibles aux instabilités, défaut dont ne souffrent pas les méthodes symboliques. En outre, ces méthodes ne sont pas limitées à des systèmes à coefficients numériques.

Maple propose plusieurs méthodes de résolution à l'utilisateur. Le choix de la méthode dépend de la forme du système et du nombre de résolutions à effectuer.

La méthode la plus simple consiste à utiliser la fonction `solve/linear`, qui choisit une stratégie de résolution selon le type des coefficients du système. C'est aussi cette fonction qu'exécute `solve` lorsqu'elle est appelée avec un système linéaire.

Exemple 7. Voici un petit exemple à trois équations et trois inconnues :

```
sys:={x+3*a*z=3,b*y+c*z=2,a*x+3*y=c}:
readlib(`solve/linear`)(sys,{x,y,z});
```

$$\left\{ z = \frac{-bc + 6 + 3ba}{3\left(c + ba^2\right)}, y = \frac{6a^2 - 3ac + c^2}{3\left(c + ba^2\right)}, x = \frac{-6a + 3c + abc}{c + ba^2} \right\}$$

Lorsque le système est donné sous forme matricielle, on appelle la fonction `linsolve` qui à son tour va appeler `solve/linear`.

Exemple 8. Voici un système $4 \times 4$ donné directement sous forme matricielle :

```
A:=matrix([[-6,-2*t-6,-t-3,-t-3],[-2*t-6,-6,-t-3,-t-3],
           [t+3,t+3,-3,2*t+6],[t+3,t+3,2*t+6,-3]]);
```

$$A := \begin{bmatrix} -6 & -2t-6 & -t-3 & -t-3 \\ -2t-6 & -6 & -t-3 & -t-3 \\ t+3 & t+3 & -3 & 2t+6 \\ t+3 & t+3 & 2t+6 & -3 \end{bmatrix}$$

```
b:=vector([t,-t,1,4]);
```

$$b := \begin{bmatrix} t & -t & 1 & 4 \end{bmatrix}$$

```
linsolve(A,b);
```

TABLE 4 Principaux calculs sur les matrices.

| Fonction | Commande |
|---|---|
| Trace | `trace` |
| Rang | `rank` |
| Noyau | `kernel` |
| Déterminant | `det` |
| Mineur | `minor` |
| Inverse | `inverse` |
| Polynôme caractéristique | `charpoly` |
| Polynôme minimal | `minpoly` |
| Valeurs propres | `eigenvals` |
| Vecteurs propres | `eigenvects` |
| Forme de JORDAN | `jordan` |
| Exponentielle | `exponential` |

$$\left[-\frac{2t+15}{6t}, -\frac{8t+15}{6t}, \frac{5t^2+57t+135}{3(9+2t)t}, \frac{5t^2+48t+135}{3(9+2t)t}\right]$$

Une troisième méthode, que l'on n'emploie jamais en calcul numérique, consiste à calculer l'inverse de la matrice $A$ avant de réaliser le produit $A^{-1}b$. Là encore, la procédure de calcul d'inverse détermine d'abord le type des éléments de la matrice avant de décider de la méthode à appliquer. Le principal intérêt de ce calcul relativement coûteux est qu'il réduit les résolutions ultérieures à un simple produit de matrices. Cette méthode est donc préférable dans le cas de nombreuses résolutions correspondant à différents vecteurs $b$.

EXEMPLE 9. Sur le même système que précédemment, le calcul s'effectue ainsi :

```
IA:=evalm(A^(-1));
```

$$IA := \begin{bmatrix} -\frac{1}{6} & -\frac{t+3}{6t} & -\frac{t+3}{6t} & -\frac{t+3}{6t} \\ -\frac{t+3}{6t} & -\frac{1}{6} & -\frac{t+3}{6t} & -\frac{t+3}{6t} \\ \frac{t+3}{6t} & \frac{t+3}{6t} & \frac{9t+27+t^2}{3t(9+2t)} & \frac{12t+27+t^2}{3t(9+2t)} \\ \frac{t+3}{6t} & \frac{t+3}{6t} & \frac{12t+27+t^2}{3t(9+2t)} & \frac{9t+27+t^2}{3t(9+2t)} \end{bmatrix}$$

```
evalm(IA &* b);
```

$$\left[-\frac{2t+15}{6t}, -\frac{8t+15}{6t}, \frac{5t^2+57t+135}{3(9+2t)t}, \frac{5t^2+48t+135}{3(9+2t)t}\right]$$

**2.2. Calculs sur des matrices.** La plupart des opérations classiques sur les matrices sont disponibles dans les systèmes de calcul formel. Le tableau 4 montre le nom des commandes Maple correspondantes. Dans ce tableau, les opérations sont regroupées en deux catégories. Les opérations de la première catégorie n'effectuent que des calculs dans le corps auquel appartiennent les coefficients de la matrice, et sont donc généralement peu coûteuses. La seconde catégorie peut nécessiter des calculs dans des extensions algébriques du

corps de base, déterminées par le polynôme caractéristique. Sauf cas particulier, ceci limite l'utilisation de ces dernières commandes à des matrices de petite taille.

Nous allons maintenant montrer quelques exemples d'utilisation de certaines de ces fonctions, et éventuellement détailler leurs limitations.

### 2.2.1. *Noyau*

La détermination du noyau d'une matrice consiste à résoudre un système linéaire homogène. Il n'y a donc là rien de plus que ce que nous avons déjà vu.

EXEMPLE 10. Pour déterminer une relation de dépendance entre vecteurs, il suffit de calculer le noyau de la matrice ayant ces vecteurs pour colonnes. Sachant que les trois vecteurs

```
v1:=vector([-86,30,80,72]):v2:=vector([66,-29,-91,-53]):
v3:=vector([-103,18,7,93]):
print(v1,v2,v3);
```

$$\begin{bmatrix}-86\\30\\80\\72\end{bmatrix}, \begin{bmatrix}66\\-29\\-91\\-53\end{bmatrix}, \begin{bmatrix}-103\\18\\7\\93\end{bmatrix}$$

sont liés, la relation de liaison s'obtient par :

```
kernel(augment(v1,v2,v3));
```

$$\{[7/6, 1, -1/3]\}$$

Autrement dit

$$7v_1 + 6v_2 - 2v_3 = 0.$$

Dans le cas général, cette méthode fournit une base des relations de liaison.

### 2.2.2. *Déterminant*

Un système de calcul formel peut calculer le déterminant de matrices symboliques jusqu'à l'ordre vingt environ, et de matrices numériques jusqu'à l'ordre cent.

EXEMPLE 11. Pour le calcul de déterminants de matrices dont la dimension est variable, le calcul formel ne peut servir qu'à faire des expériences. Pour calculer les déterminants de matrices tridiagonales de la forme

$$\begin{vmatrix}1+x^2 & -x & 0 & 0\\ -x & 1+x^2 & -x & 0\\ 0 & -x & 1+x^2 & -x\\ 0 & 0 & -x & 1+x^2\end{vmatrix}$$

nous définissons une procédure qui produit la matrice d'ordre $n$ et nous calculons le déterminant pour diverses valeurs de $n$.

```
genm:=n->band([-x,1+x^2,-x],n):
det(genm(9)), det(genm(10));
```

$$1 + x^2 + x^4 + x^6 + x^8 + x^{10} + x^{12} + x^{14} + x^{16} + x^{18},$$
$$1 + x^2 + x^4 + x^6 + x^8 + x^{10} + x^{12} + x^{14} + x^{16} + x^{18} + x^{20}$$

Les valeurs pour $n = 9$ et $n = 10$ nous permettent de deviner la formule $\sum_{i=0}^{n} x^{2i}$. Pour prouver cette formule, il suffit de développer le déterminant par rapport à la dernière colonne ce qui fournit une récurrence. Pour le développement Maple ne nous est d'aucun secours car il ne sait pas manipuler des matrices dont la dimension est un paramètre ; en revanche il saura résoudre la récurrence linéaire obtenue (voir exercice 2 p. 132).

### 2.2.3. *Polynôme caractéristique*

Le calcul de déterminants symboliques permet celui de polynômes caractéristiques. La connaissance de ce polynôme fournit de nombreuses informations sur la matrice.

Exemple 12. Nous reprenons la matrice $A$ de l'exemple 8 p. 121. Le calcul de son polynôme caractéristique est très facile (avec un système de calcul formel !) :

```
charpoly(A,lambda);
```

$$\lambda^4 + 18\lambda^3 - 24\lambda^2 t - 36\lambda t^2 + 81\lambda^2 - 216\lambda t + 108t^2 - 4\lambda^2 t^2 + 24t^3$$

Nous calculons alors le discriminant de ce polynôme, et même sa décomposition sans carrés, ce qui est peu coûteux (voir chap. VI) :

```
convert(discrim(",lambda),sqrfree);
```

$$768(8t + 27)(4t + 9)^2 \left(t^2 + 3t\right)^4$$

De cela il découle que pour $t \notin \{-27/8, -9/4, 0, -3\}$ le polynôme caractéristique n'a que des racines simples éventuellement dans une extension algébrique de $\mathbb{Q}(t)$, et donc que la matrice est diagonalisable pour toutes ces valeurs de $t$.

### 2.2.4. *Valeurs propres*

Le calcul des valeurs et vecteurs propres d'une matrice, à coefficients symboliques ou non, revient à trouver les racines du polynôme caractéristique puis à résoudre un système linéaire. Les systèmes de calcul formel sont donc restreints dans ce domaine à leur capacité à résoudre de telles équations (voir chap. VI).

Exemple 13. Toujours sur la même matrice, le calcul ne pose pas de problème :

```
eigenvals(A);
```

$$2t, -9/2 - \frac{\sqrt{81 + 24t}}{2}, -9/2 + \frac{\sqrt{81 + 24t}}{2}, -9 - 2t$$

Lorsque les valeurs propres ne sont pas connues explicitement, il est parfois possible de poursuivre le calcul avec des nombres ou fonctions algébriques grâce à la notation `RootOf` (voir chap. VI).

EXEMPLE 14. Voici ce qui se produit le plus souvent (`randmatrix` produit une matrice aléatoire) :

```
B:=randmatrix(5,5);
```

$$\begin{bmatrix} -17 & -98 & -36 & 40 & 22 \\ 5 & -88 & -43 & -73 & 25 \\ 4 & -59 & 62 & -55 & 25 \\ 9 & 40 & 61 & 40 & -78 \\ 62 & 11 & 88 & 1 & 30 \end{bmatrix}$$

```
eigenvals(B);
```

$$\text{RootOf}\left(-4512860849 + 39920068\_Z - 74744\_Z^2 - 6573\_Z^3 - 27\_Z^4 + \_Z^5\right)$$

S'il n'est pas nécessaire de distinguer entre les différentes racines de ce polynôme, alors le calcul se poursuit généralement sans trop de difficulté, sinon il s'agit d'un problème qui est à la limite des possibilités actuelles du calcul formel.

### 2.2.5. *Vecteurs propres*

Connaissant les valeurs propres, les vecteurs propres s'obtiennent par résolution d'un système linéaire. La difficulté supplémentaire est que cette résolution s'effectue avec des matrices dans le corps engendré par les entrées de la matrice de départ *et par les valeurs propres*, ce qui peut être coûteux.

EXEMPLE 15. Pour la première matrice (p. 121) nous obtenons :

```
eigenvects(A);
```

$$\left[\%_1, 1, \left\{\left[-\frac{2t+3-\%_1}{2(t+3)}, -\frac{2t+3-\%_1}{2(t+3)}, 1, 1\right]\right\}\right],$$
$$[-2t-9, 1, \{[0,0,-1,1]\}], [2t, 1, \{[-1,1,0,0]\}]$$
$$\%_1 = \text{RootOf}\left(-6t + \_Z^2 + 9\_Z\right)$$

Ce résultat doit s'interpréter ainsi : il s'agit d'une suite de listes, chaque liste comprenant une valeur propre, sa multiplicité, et une base de vecteurs propres correspondant à cette valeur propre. Dans le cas des deux dernières valeurs propres, ce résultat est assez clair. Dans le cas de la première cependant, il faut interpréter le résultat en faisant prendre à cette valeur propre chacune des racines du polynôme $z^2 + 9z - 6t$ comme valeur, ce qui donne bien tous les vecteurs propres. Dans ce cas simple, comme nous savons que le polynôme caractéristique se résout explicitement, la solution s'obtient par

```
eigenvects(A,radical);
```

et est trop longue pour être reproduite ici.

EXEMPLE 16. Pour la seconde matrice, `eigenvects(B);` renvoie un résultat trop long pour être reproduit ici qui consiste en une valeur propre (celle de l'exemple 14) de multiplicité 1 et une base constituée d'un vecteur propre dont les coordonnées sont exprimées en termes de cette valeur propre. Ce résultat s'interprète comme précédemment.

2.2.6. *Forme de* JORDAN

Il est parfois utile de *diagonaliser* une matrice. Cela fournit une base dans laquelle l'endomorphisme associé se comporte d'une façon agréable. La diagonalisation n'est pas toujours possible (il faut qu'il existe une base de vecteurs propres). La forme de JORDAN d'une matrice à coefficients complexes existe toujours et est diagonale quand la matrice est diagonalisable. Le calcul de cette forme requiert la détermination des vecteurs propres et rencontre donc les difficultés mentionnées ci-dessus.

EXEMPLE 17. Notre première matrice ne pose pas de problème :

```
jordan(A);
```

$$\begin{bmatrix} -\frac{9}{2}-\frac{1}{2}\sqrt{81+24t} & 0 & 0 & 0 \\ 0 & -\frac{9}{2}+\frac{1}{2}\sqrt{81+24t} & 0 & 0 \\ 0 & 0 & 2t & 0 \\ 0 & 0 & 0 & -2t-9 \end{bmatrix}$$

En revanche, le calcul de la matrice de passage ne parvient pas à son terme :

```
jordan(A,'P');
Error, (in linalg/frobenius/constpas)
unable to execute for statement
```

Il s'agit là d'une faiblesse de la version actuelle de Maple.

EXEMPLE 18. Sur la seconde matrice `jordan` renvoie une matrice diagonale qui semble multiple de l'identité. Il faut cependant interpréter les éléments de la diagonale comme tous différents. De manière surprenante, le calcul de la matrice de changement de base s'effectue sans problème.

2.2.7. *Polynômes et séries de matrices*

Pour calculer un polynôme en une matrice, il suffit d'utiliser `evalm` comme nous l'avons déjà vu. En particulier le calcul d'une puissance $k^{\text{e}}$ d'une matrice est assez rapide car la méthode d'exponentiation binaire (voir IV§1.1.1) est employée.

Il est parfois nécessaire de calculer non pas des polynômes en une matrice, mais des séries. Ce calcul est possible, et fournit une application importante de la diagonalisation et de la réduction en forme de JORDAN.

Nous allons maintenant en donner deux applications.

*Itérés fractionnaires.* Si $A$ est la matrice associée à un endomorphisme $f$, la matrice $A^2$ est associée à $f \circ f$. Inversement on peut se poser la question de trouver un endomorphisme $g$ tel que $g \circ g = f$. Si $B$ est la matrice associée, on doit alors avoir $A = B^2$ ou autrement dit $B = \pm A^{1/2}$.

Pour calculer cette racine carrée, on a recours au développement en série de $\sqrt{1+z}$ : si $|z| < 1$ alors

$$\sqrt{1+z} = 1 - \frac{z}{2} - \frac{z^2}{8} + \cdots$$

Il nous suffit d'écrire la matrice $A$ sous la forme $I + (A - I)$ et d'appliquer formellement le développement précédent. La condition $|z| < 1$ est remplacée par la même condition sur le *rayon spectral* de la matrice, c'est-à-dire le maximum des modules des valeurs propres.

Le calcul de la racine carrée est ainsi réduit à des calculs d'additions et de puissances entières de matrices. Si la matrice $A$ est diagonale, tout se passe bien : chaque élément de la diagonale peut être traité indépendamment des autres. Il s'ensuit que $B$ est simplement la matrice dont la diagonale contient les racines carrées des éléments de $A$. Si $A$ est diagonalisable, c'est-à-dire qu'il existe une matrice de changement de base $P$ telle que $A = PDP^{-1}$, où $D$ est diagonale alors il suffit d'effectuer le calcul ci-dessus sur $D$ et d'effectuer à la fin le produit (facile) $P^{-1}(I + (D - I))^{1/2}P$.

Reste le cas où la matrice n'est pas diagonalisable. Alors la même technique s'applique à la forme de JORDAN, forme dans laquelle le calcul des puissances permet de traiter les différentes entrées de la matrice indépendamment les unes des autres.

Les problèmes que nous avions mentionnés plus haut pour la détermination des vecteurs propres se posent bien entendu aussi ici, puisque nous passons par le calcul de la forme de JORDAN.

EXEMPLE 19. La matrice suivante n'est pas diagonalisable :

```
A:=matrix([[0,3,-1],[1,-1,2],[1,0,1]]);
```

$$A := \begin{bmatrix} 0 & 3 & -1 \\ 1 & -1 & 2 \\ 1 & 0 & 1 \end{bmatrix}$$

Voici sa forme de JORDAN et la matrice de changement de base correspondante :

```
d:=jordan(A,'P'): print(d,P);
```

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix}, \qquad \begin{bmatrix} 1/2 & 1/2 & 1/2 \\ -1/3 & 0 & 1/3 \\ 0 & -1 & 1 \end{bmatrix}$$

Pour calculer la racine carrée il nous faut les puissances de $(d - I)$. Les premières puissances permettent de deviner la forme générale, qui n'est pas difficile à établir par récurrence :

```
evalm(d-1),evalm((d-1)^2),evalm((d-1)^3),evalm((d-1)^4);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -2 & 1 \\ 0 & 0 & -2 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & -4 \\ 0 & 0 & 4 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & -8 & 12 \\ 0 & 0 & -8 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 16 & -32 \\ 0 & 0 & 16 \end{bmatrix}$$

D'où

$$(d - I)^p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & (-2)^p & -p(-2)^{p-1} \\ 0 & 0 & (-2)^p \end{bmatrix}.$$

Donc pour toute série $f$ dont le rayon de convergence est supérieur à 2,

$$f(d-I)=\begin{bmatrix} f(1) & 0 & 0 \\ 0 & f(-2) & -f'(-2) \\ 0 & 0 & f(-2) \end{bmatrix}.$$

L'application de cette formule à $f=\sqrt{1+z}$ (dont la série a pourtant un rayon de convergence inférieur à 2) nous fournit une matrice :

$$B=\begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & i & i/2 \\ 0 & 0 & i \end{bmatrix}$$

dont nous vérifions que le carré redonne $d$ :

```
B:=matrix([[sqrt(2),0,0],[0,I,-I/2],[0,0,I]]):evalm(B^2);
```

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix}$$

Il ne reste plus qu'à revenir dans la base de départ où l'identité cherchée est bien sûr vérifiée :

```
F:=evalm(P^(-1) &* B &* P):print(F,evalm(F^2));
```

$$\begin{bmatrix} \frac{\sqrt{2}+2i}{3} & \frac{\sqrt{2}-4i}{3} & \frac{\sqrt{2}+2i}{3} \\ \frac{\sqrt{2}-i}{3} & \frac{2\sqrt{2}+7i}{6} & \frac{2\sqrt{2}-5i}{6} \\ \frac{\sqrt{2}-i}{3} & \frac{2\sqrt{2}+i}{6} & \frac{2\sqrt{2}+i}{6} \end{bmatrix}, \quad \begin{bmatrix} 0 & 3 & -1 \\ 1 & -1 & 2 \\ 1 & 0 & 1 \end{bmatrix}$$

*Exponentielle d'une matrice.* Cette seconde application des séries de matrices est très importante : c'est ainsi que l'on résout les systèmes différentiels linéaires à coefficients constants. En effet, un tel système s'écrit

$$\frac{dX(t)}{dt}=AX(t), \qquad X(0) \text{ donné},$$

où $X(t)$ est un vecteur $(x_1(t),\dots,x_n(t))$ d'ordre $n$ et $A$ est une matrice $n\times n$ indépendante de $t$. L'unique solution de ce système est donnée par

$$X(t)=e^{tA}X(0).$$

Comme ci-dessus, l'exponentielle de la matrice est définie par son développement en série :

$$\exp(A)=\sum_{k=0}\frac{A^k}{k!}$$

avec $A^0=I$ la matrice identité.

Exemple 20. La matrice de l'exemple précédent correspond au système :

$$\begin{cases} x'(t)=3y(t)-z(t), \\ y'(t)=x(t)-y(t)+2z(t), \\ z'(t)=x(t)+z(t). \end{cases}$$

D'après ce que nous avons vu dans l'exemple précédent, pour toute série $f$ de rayon de convergence supérieur à t,

$$f(td) = \begin{bmatrix} f(2t) & 0 & 0 \\ 0 & f(-t) & -tf'(-t) \\ 0 & 0 & f(-t) \end{bmatrix}.$$

Il ne nous reste plus qu'à faire le changement de base inverse :

```
B:=matrix([[exp(2*t),0,0],[0,exp(-t),t*exp(-t)],[0,0,exp(-t)]]):
F:=evalm(P^(-1) &* B &* P);
```

$$F := \begin{bmatrix} e^{2t}/3 + 2e^{-t}/3 & e^{2t}/3 + 2te^{-t} - e^{-t}/3 & e^{2t}/3 - e^{-t}/3 - 2te^{-t} \\ e^{2t}/3 - e^{-t}/3 & e^{2t}/3 - te^{-t} + 2e^{-t}/3 & e^{2t}/3 - e^{-t}/3 + te^{-t} \\ e^{2t}/3 - e^{-t}/3 & e^{2t}/3 - te^{-t} - e^{-t}/3 & e^{2t}/3 + 2e^{-t}/3 + te^{-t} \end{bmatrix}$$

matrice qui donne la solution du système différentiel.

Pour le cas de l'exponentielle, il n'est pas nécessaire d'effectuer soi-même toutes les étapes intermédiaires : Maple fournit la commande `exponential` qui se charge de tout.

**2.3. Optimisation linéaire.** Un problème d'optimisation linéaire consiste à minimiser (ou maximiser) une fonction linéaire sous contraintes linéaires. Un tel problème peut toujours se mettre sous la forme standard suivante :

$$(\mathcal{P}) \quad \begin{cases} \min z = {}^t cx, \\ Ax = b, \\ x \geq 0, \end{cases}$$

où $z$ est la fonction à minimiser (fonction objectif ou fonction économique), $c$ est le vecteur des coûts, de dimension $n$, $n$ est le nombre de variables, $m$ est le nombre de contraintes (plus petit que $n$), $A$ est une matrice réelle $m \times n$ de rang $m$ (matrice des contraintes), et $b$ est le vecteur des seconds membres, de dimension $m$.

L'expression $\min z = {}^t cx$ signifie que l'on cherche le vecteur $x$ qui minimise la forme linéaire ${}^t cx$, $c$ étant donné. Nombreux sont les problèmes pratiques qui se ramènent à cette forme.

La méthode classique pour résoudre ce problème est la méthode du simplexe dont le principe est assez simple. En gros le problème est réduit à parcourir des sommets d'un polyèdre convexe (dans un espace dont la dimension peut-être élevée) en suivant ses arêtes. Une description complète et détaillée de l'algorithme du simplexe dépasse le cadre de cet ouvrage. En particulier, il est parfois difficile de trouver un premier sommet où démarrer l'algorithme, et des cas de dégénérescence se produisent lorsque l'une des face du polyèdre est parallèle à l'hyperplan noyau de la forme à minimiser.

Il existe en Maple un *package* appelé `simplex` qui donne directement les solutions d'un tel problème.

Exemple 21. Comme illustration, nous allons utiliser l'exemple suivant :

$$\begin{cases} \min z = -x_1 - 2x_2, \\ -3x_1 + 2x_2 + x_3 = 2, \\ -x_1 + 2x_2 + x_4 = 4, \\ x_1 + x_2 + x_5 = 5, \\ x_1, x_2, x_3, x_4, x_5 \geq 0. \end{cases}$$

La résolution est immédiate :

```
constr:={-3*x[1]+2*x[2]+x[3]=2,
         -x[1]+2*x[2]+x[4]=4,x[1]+x[2]+x[5]=5}:
simplex[minimize](-x[1]-2*x[2],constr,NONNEGATIVE);
```

$$\{x_1 = 2, x_2 = 3, x_4 = 0, x_5 = 0, x_3 = 2\}$$

Le *package* `simplex` comprend en plus des fonctions exécutant les étapes de l'algorithme, permettant ainsi l'apprentissage, l'expérimentation et l'orientation de la méthode du simplexe.

**2.4. Automatique.** Les applications du calcul formel sont importantes dans le domaine de l'automatique comme nous le verrons plus en détail dans le chapitre XII. Un grand nombre de systèmes dynamiques en automatique sont décrits par :

$$\begin{cases} \dot{X} = f(X, U), \\ Y = g(X, U), \end{cases}$$

où $X$, $Y$ et $U$ sont des vecteurs représentant l'état, la sortie et la commande du système. $\dot{X}$ représente la dérivée de $X$ par rapport au temps. Lorsque $f$ et $g$ ne sont pas linéaires, il est souvent utile d'obtenir le système correspondant linéarisé autour d'un point de fonctionnement $(X_0, U_0)$ :

$$\begin{cases} \dot{X} = AX + BU, \\ Y = CX + DU, \end{cases}$$

où $A$, $B$, $C$ et $D$ sont les matrices jacobiennes de $f$ et $g$ par rapport à $X$ et $U$ en $(X_0, U_0)$. (La matrice jacobienne du vecteur $(f_1(X), \ldots, f_m(X))$ par rapport à $(X_1, \ldots, X_n)$ est la matrice dont l'élément $(i, j)$ est $\partial f_i / \partial X_j$.)

Pour fixer les idées voici les calculs mis en jeu pour le système consistant en un pendule inversé sur un chariot mobile (fig. 1 p. 131). Les équations du pendule sont les suivantes :

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = [u_1 + \sqrt{m_b}(\sin(x_3)x_4^2 - \cos(x_3)D_4)]/(m_b + m_c) \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = D_4 \end{cases} \qquad \begin{cases} y_1 = x_1 \\ y_2 = x_3 \end{cases}$$

avec

$$\begin{cases} D_4 &= -\frac{1}{d}[\sin(x_3)\cos(x_3)q_m x_4^2 + \frac{2}{dm_b l}\sin(x_3)m_b g - q_m \cos(x_3)u_1] \\ q_m &= m_b/(m_b + m_c) \\ d &= 4/3 - q_m \cos(x_3)^2 \end{cases}$$

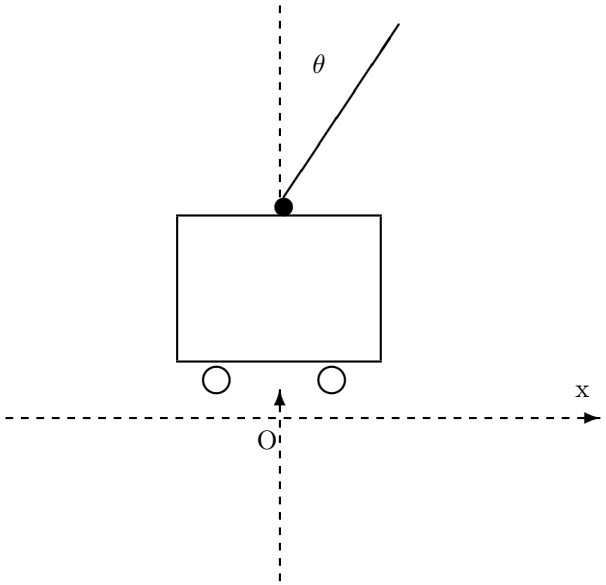



FIGURE 1 Le pendule inversé sur un chariot mobile.

où $x_1$, $x_2$, $x_3$ et $x_4$ correspondent respectivement à la position du chariot, à sa vitesse, à l'angle $\theta$ du pendule avec la verticale et à sa vitesse angulaire, $m_b$ et $m_c$ sont respectivement les masses du pendule et du chariot et $l$ est la longueur du pendule. La commande $u_1$ représente la force avec laquelle le chariot est poussé ou tiré.

Le calcul en Maple donne :

```
x:=vector(4): u:=vector(1):
qm:=mb/(mb+mc): c3:=cos(x[3]): s3:=sin(x[3]): d:=4/3-qm*c3^2:
D4:=(-s3*c3*qm*x[4]^2+2/(mb*l)*(s3*mb*g-qm*c3*u[1]))/d:
f:=vector([x[2],(u[1]+mb^(1/2)*(s3*x[4]^2-c3*D4))/(mb+mc),
           x[4],D4]):
g:=vector([x[1],x[3]]):ini:=[seq(x[i]=0,i=1..4),u[1]=0]:
seq(map(simplify,i),
      i=subs(ini,zip(jacobian,[f,f,g,g],[x,u,x,u])));
```

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -3\dfrac{gmb}{mb+4mc} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 6\dfrac{g(mb+mc)}{l(mb+4mc)} & 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 4\dfrac{1}{mb+4mc} \\ [2ex]0 \\ -6\dfrac{1}{l(mb+4mc)} \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Le calcul est très rapidement réalisé et les matrices peuvent être maintenant utilisées pour d'autres opérations, en particulier pour la commande du chariot, par exemple pour que le pendule reste vertical.

**2.5. Exercices.**

1. Écrire une procédure prenant en entrée une matrice $M$ et un numéro de colonne $j$, et calculant le déterminant de $M$ par la formule des cofacteurs. Vérifier le résultat sur une matrice générique d'ordre 6.
2. On cherche à calculer une formule générale pour le déterminant de taille $n$
$$\begin{vmatrix} x & 1 & 0 & \cdots & 0 \\ 1 & x & 1 & \ddots & \vdots \\ 0 & 1 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & x & 1 \\ 0 & \cdots & 0 & 1 & x \end{vmatrix}.$$
Trouver une formule de récurrence, la résoudre avec `rsolve`, simplifier la formule obtenue, puis vérifier le résultat pour $n = 6$.
3. Déterminer les puissances et la racine carrée de
$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}.$$
4. La matrice $A$ et les vecteurs $b$ et $c$ du §2.3 s'obtiennent directement à partir de l'expression de $z$ et des contraintes. Écrire une procédure Maple qui réalise ce travail.

## 3. Espaces vectoriels euclidiens

L'espace ambiant est euclidien. Les solides sont rigides et leur mouvement est régi par des isométries. Outre les opérations générales de l'algèbre linéaire que nous avons déjà vues, on y dispose d'un produit scalaire (`dotprod`), ce qui permet de calculer des angles ou de déterminer des bases orthogonales (`GramSchmidt`). Là encore, les calculs manuels sont rapidement infaisables mais restent à la portée des systèmes de calcul formel.

**3.1. Isométries.** Pour illustrer quelques calculs dans l'espace euclidien, nous allons calculer la matrice d'une rotation de l'espace à trois dimensions donnée par un vecteur directeur $\vec{u}$ unitaire de son axe et son angle $\alpha$ mesuré en utilisant l'orientation définie par $\vec{u}$. Un vecteur quelconque $\vec{v}$ est transformé par cette rotation en

$$\cos\alpha\vec{v} + \sin\alpha(\vec{u}\wedge\vec{v}) + (1-\cos\alpha)(\vec{u}\cdot\vec{v})\vec{u}$$

où $\wedge$ représente le produit vectoriel et $\cdot$ le produit scalaire.

En Maple cela donne :

```
u:=vector([p,q,r]):
basis:=vector([1,0,0]),vector([0,1,0]),vector([0,0,1]):
R:=augment(seq(evalm(cos(alpha)*v+sin(alpha)*crossprod(u,v)
   +(1-cos(alpha))*dotprod(u,v)*u),v=basis)):
```

Nous ne montrons que la première colonne :

$$\begin{bmatrix} \cos(\alpha)+(1-\cos(\alpha))p^2 \\ \sin(\alpha)r+(1-\cos(\alpha))pq \\ -\sin(\alpha)q+(1-\cos(\alpha))pr \end{bmatrix}.$$

Inversement, si l'on connaît la matrice de rotation, son axe et son angle peuvent se retrouver grâce à deux propriétés que nous allons démontrer à l'aide de Maple.

La première propriété concerne la trace, qui est égale à $2\cos\alpha+1$. Nous commençons par créer une règle de simplification qui précise que le vecteur $\vec{u}$ ci-dessus était unitaire :

```
eq:={p^2+q^2+r^2=1}:
simplify(trace(R),eq);
```

$$2\cos\alpha+1$$

La seconde propriété utile s'obtient en calculant $R-{}^tR$ : cette matrice est antisymétrique de la forme

$$2\sin\alpha\begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix}.$$

Nous le prouvons avec Maple :

```
evalm(R-transpose(R));
```

$$\begin{bmatrix} 0 & -2\sin(\alpha)r & 2\sin(\alpha)q \\ 2\sin(\alpha)r & 0 & -2\sin(\alpha)p \\ -2\sin(\alpha)q & 2\sin(\alpha)p & 0 \end{bmatrix}$$

En dimension 3, on utilise souvent les coordonnées sphériques. Le vecteur unitaire $\vec{u}$ s'écrira alors plutôt $(\sin\phi\cos\theta,\sin\phi\sin\theta,\cos\phi)$. Les calculs deviennent rapidement infaisables à la main comme le montre la première colonne de la matrice $R$ :

```
sph:=[p=sin(phi)*cos(theta),q=sin(phi)*sin(theta),r=cos(phi)]:
col(subs(sph,op(R)),1);
```

$$\begin{bmatrix} \cos\alpha+(1-\cos\alpha)\sin^2\phi\cos^2\theta \\ \sin\alpha\cos\phi+(1-\cos\alpha)\sin^2\phi\cos\theta\sin\theta \\ (1-\cos\alpha)\sin\phi\cos\theta\cos\phi-\sin\alpha\sin\phi\sin\theta \end{bmatrix}.$$

Même si l'on connaît l'axe et que la seule inconnue est l'angle $\alpha$, l'expression reste compliquée, surtout si l'on se met à composer des rotations d'axes différents.

**3.2. Réduction d'une forme quadratique.** Un polynôme du second degré en $n$ variables réelles peut s'écrire

$$\Phi : \quad v\in\mathbb{R}^n \longrightarrow {}^tvAv+{}^tbv+c\in\mathbb{R}$$

où $A$ est une matrice symétrique $n\times n$, $b$ est un vecteur de $\mathbb{R}^n$ et $c$ est un réel. La partie homogène du second degré ${}^tvAv$ est une forme quadratique. Sa réduction consiste à trouver une base orthonormée dans laquelle la matrice $A$ est diagonale. C'est toujours possible puisque $A$ est symétrique.

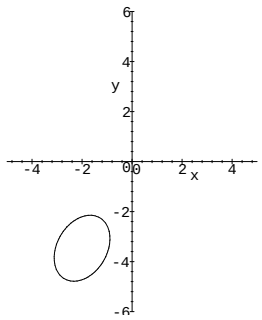



FIGURE 2 La conique $\frac{7}{8}x^2 - \frac{\sqrt{3}}{4}xy + \frac{5}{8}y^2 + 2x + 2\sqrt{3}y + 7 = 0$.

Si l'on se place en dimension 2, $\Phi(x, y) = 0$ est l'équation d'une conique dans un repère orthonormé $(\overrightarrow{Ox}, \overrightarrow{Oy})$. Maple va nous permettre de visualiser ce que réduire la forme quadratique signifie pour la conique associée. La conique considérée (représentée fig. 2) est définie par l'équation :

$$\frac{7}{8}x^2 - \frac{\sqrt{3}}{4}xy + \frac{5}{8}y^2 + 2x + 2\sqrt{3}y + 7 = 0$$

ce qui correspond à

```
A:=matrix([[7/8,-sqrt(3)/8],[-sqrt(3)/8,5/8]]):
b:=vector([2,2*sqrt(3)]): c:=7:
```

Le polynôme étant quant à lui défini par

```
v:=vector(2):
Phi:=transpose(v) &* A &* v +transpose(b) &* v + c:
```

Nous allons dans un premier temps chercher une base orthonormée dans laquelle la matrice $A$ est diagonale. Il suffit pour cela de calculer la forme de JORDAN de la matrice $A$ (qui est ici diagonale) et de normaliser les lignes de la matrice de changement de base :

```
jordan(A,'P');
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1/2 \end{bmatrix}$$

```
P:=transpose(stack(seq(map(simplify,normalize(i)),
   i=row(P,1..2))));
```

$$\begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} \\ \frac{-1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix}$$

Dans cette base, le polynôme devient :

```
evalm(subs(v=evalm(P &* vector([x,y])),Phi));
```

$$x^2 + \frac{1}{2}y^2 + 4y + 7$$

(voir fig. 3).

Les termes en $xy$ ont bien disparu, et les coefficients de $x^2$ et $y^2$ sont les valeurs propres. Nous avons donc bien réduit la forme quadratique. Les

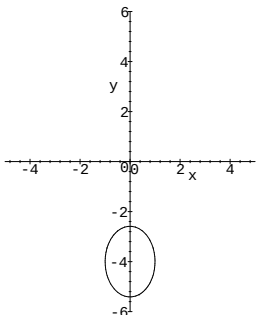



FIGURE 3 La conique réduite $x^2 + \frac{1}{2}y^2 + 4y + 7 = 0$.

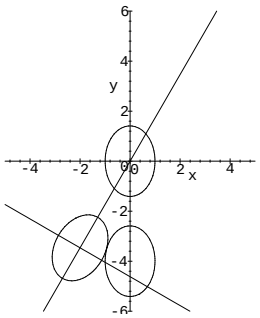



FIGURE 4 Réduction d'une conique.

termes en $x$ et $y$ peuvent encore être supprimés par une translation qui ramène le centre $(-4, 0)$ de la conique à l'origine.

```
expand(subs([x=X,y=-4+Y],"));
```

$$X^2 + \frac{1}{2}Y^2 - 1$$

Pour conclure, nous donnons une interprétation géométrique. La matrice $P$ est une matrice de rotation d'angle $-\pi/6$ :

$$P = \begin{bmatrix} \cos(\pi/6) & \sin(\pi/6) \\ -\sin(\pi/6) & \cos(\pi/6) \end{bmatrix}.$$

Les axes principaux de la conique sont donc inclinés de $-\pi/6$ par rapport au repère $(\overrightarrow{Ox}, \overrightarrow{Oy})$. La transformation de matrice $P$ permet de prendre pour directions principales de la conique les axes du repère et la translation ramène ensuite le centre de la conique à l'origine. Les trois coniques obtenues durant la réduction sont récapitulées en figure 4.

Les mêmes calculs s'appliquent bien sûr pour un ordre plus élevé. En dimension 3 on obtient des surfaces quadriques (ellipsoïdes, paraboloïdes,...).

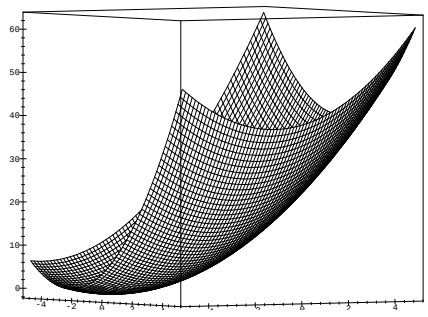


FIGURE 5 Le paraboloïde

$$z = \frac{7}{8}x^2 - \frac{\sqrt{3}}{4}xy + \frac{5}{8}y^2 + 2x + 2\sqrt{3}y + 7.$$

Pour les dimensions supérieures la visualisation des résultats devient plus difficile.

**3.3. Optimisation quadratique.** Un problème relié au précédent est celui de la minimisation d'une forme quadratique. Il s'agit de trouver un vecteur $v$ qui minimise $\Phi(v)$.

Tout d'abord, pour que ce minimum existe, il faut que les valeurs propres de la matrice $A$ soient positives ou nulles, ce que nous supposerons par la suite. Le vecteur gradient de $\Phi$ par rapport à $v$ est égal à $2Av + b$ (car $A$ est symétrique). Le minimum de $\Phi(v)$ est atteint lorsque le gradient est nul, donc pour $v = -A^{-1}b/2$ à condition que $A$ soit inversible.

EXEMPLE 22. La surface

$$z = \frac{7}{8}x^2 - \frac{\sqrt{3}}{4}xy + \frac{5}{8}y^2 + 2x + 2\sqrt{3}y + 7$$

est un paraboloïde de révolution (fig. 5). Le minimum est facilement déterminé par la commande `linsolve` :

```
linsolve(evalm(2*A),-b);
```

$$\left[\begin{array}{cc} -2 & -2\sqrt{3} \end{array}\right]$$

Ce point n'est autre que le centre de la conique trouvé dans la réduction de la forme quadratique du paragraphe précédent. En effet, le minimum d'une forme quadratique (lorsque les valeurs propres sont positives) est atteint au centre de la conique associée. La valeur en ce point est celle du terme constant dans l'équation de la quadratique réduite (ici $-1$).

Là encore les mêmes calculs s'appliquent à un ordre plus élevé. L'avantage d'un système de calcul formel est que pour toutes ces questions, il suffit de recommencer les calculs déjà réalisés en changeant les données (ici $A$, $b$ et $c$).

### 3.4. Exercices.

1. Une condition pour que trois vecteurs $\vec{u}$, $\vec{v}$ et $\vec{w}$ soient coplanaires est que le déterminant d'ordre 3 formé par leurs coordonnées soit nul. Écrire une procédure Maple qui détermine par cette méthode si quatre points $P_1$, $P_2$, $P_3$ et $P_4$, donnés par leurs coordonnées, sont coplanaires. Vérifier que cela revient à calculer le produit mixte $(\vec{u} \wedge \vec{v}) \cdot \vec{w}$.
2. La matrice $A$, le vecteur $b$ et la constante $c$ du §3.2 s'obtiennent directement à partir de l'expression du polynôme ${}^t v A v + {}^t b v + c$. Écrire une procédure qui réalise ce travail.
3. Réaliser la réduction du polynôme défini par
$$A = \begin{bmatrix} \frac{7}{8} & -\frac{\sqrt{3}}{4} \\ 0 & \frac{5}{8} \end{bmatrix} \qquad b = \begin{bmatrix} 2 \\ 2\sqrt{3} \end{bmatrix} \qquad c = 7.$$
4. Il est possible de calculer la valeur du minimum du polynôme du second degré puisque l'on sait que $v = -A^{-1}b/2$. On trouve $c - {}^t b A^{-1} b/4$. Que devient cette valeur si $A$ n'est pas symétrique ?
5. Réduire l'ellipsoïde
$$\frac{7}{8}x^2 - \frac{3}{8}xy + \frac{\sqrt{3}}{8}xz + \frac{23}{32}y^2 + \frac{3\sqrt{3}}{16}yz + \frac{29}{32}z^2 - 2\sqrt{3}x + \sqrt{3}y - z + 3 = 0.$$
Visualiser le résultat. Trouver de deux façons différentes $x$, $y$ et $z$ pour lesquels la valeur de la forme quadratique associée à l'ellipsoïde est minimum.
6. Étudier l'existence d'un minimum de
$$f(a,b,c) = \int_0^{+\infty} (x^3 + ax^2 + bx + c)^2 e^{-2x}\, dx$$
sur $\mathbb{R}^3$. [Indication : considérer le produit scalaire
$$(P,Q) \mapsto \int_0^{+\infty} P(x)Q(x)e^{-2x}\, dx.]$$
7. Soit $f$ un endomorphisme d'un espace vectoriel euclidien de dimension 3. Sa matrice dans une base orthonormée est
$$M = -\frac{2}{3}\begin{bmatrix} -\frac{1}{2} & \frac{v}{u} & \frac{w}{u} \\ \frac{u}{v} & -\frac{1}{2} & \frac{w}{v} \\ \frac{u}{w} & \frac{v}{w} & -\frac{1}{2} \end{bmatrix}, \qquad (u,v,w) \in \mathbb{R}^{\star 3}.$$
   (1) Diagonaliser $f$. Interpréter géométriquement.
   (2) Donner une condition sur $u$, $v$, $w$ pour que $f$ soit une isométrie.

CHAPTER VI

# Polynômes et fractions rationnelles

Les polynômes sont des constituants fondamentaux du calcul formel : c'est la classe d'objets la plus simple où intervient une variable. Une fois construits les polynômes en une variable sur $\mathbb{Q}$ on peut construire les corps de nombres algébriques (cf. §1.3.8) et le corps des fractions rationnelles en une variable. Ensuite, on peut construire les polynômes à coefficients dans ces corps, et ainsi obtenir des polynômes et des fractions rationnelles à plusieurs variables. Du point de vue du calcul formel, ces constructions présentent l'avantage qu'à tout moment une *forme normale* des expressions est calculable, forme sous laquelle la nullité d'une expression et l'égalité de deux expressions sont apparentes. Cette propriété s'étend en acceptant des nombres *transcendants* (qui ne sont racine d'aucun polynôme à coefficients rationnels) à la place des variables, à condition que les nombres transcendants introduits soient algébriquement indépendants (aucun polynôme non nul à coefficients rationnels ne s'annule lorsqu'on remplace les variables par ces nombres). Ce sont ces expressions, construites uniquement par les quatre opérations élémentaires de corps $+$, $-$, $\times$, $\div$ à partir d'une ou plusieurs variables ou nombres transcendants et du corps des rationnels ou d'un corps de nombres algébriques, que l'on appelle *fractions rationnelles* dans ce chapitre.

Ainsi, $t^3 \log 2 + 1/y + z^n - z^3$ est un polynôme en $t$ et une fraction rationnelle en $t$ et $y$, mais ce n'est pas un polynôme en $z$ du point de vue du calcul formel tant qu'on n'a pas précisé une valeur entière pour $n$ ; en effet $n$ peut encore prendre n'importe quelle valeur symbolique comme par exemple $y \log t$.

On distinguera deux cas. En §1 les polynômes et fractions rationnelles sont univariés, c'est-à-dire que l'on travaille dans $K[x]$, où $K$ peut éventuellement être $\mathbb{Q}(x_1, \ldots, x_n)$, mais les variables n'interfèrent pas[†]. Puis en §2, on s'intéresse aux systèmes d'équations polynomiales en plusieurs variables et aux problèmes géométriques associés.

## 1. Opérations de base et polynômes en une variable

Le tableau 1 présente les principales opérations que l'on peut effectuer sur les polynômes et les fractions rationnelles. On dispose ainsi d'une boîte à outils d'usage très général, puisque nombreuses sont les expressions que l'on peut

[†]Plus rigoureusement, les idéaux sont tous principaux.

TABLE 1 Principales opérations sur les polynômes et fractions rationnelles.

| Type | Fonction | Commande |
|---|---|---|
| Extraction d'information syntaxique | Degré d'un polynôme | `degree` |
| | Valuation d'un polynôme | `ldegree` |
| | Coefficient | `coeff` |
| | Coefficient de tête | `lcoeff` |
| | Coefficient de queue | `tcoeff` |
| | Coefficients | `coeffs` |
| Récriture | Développer | `expand`$^\star$ |
| | Développer par rapport à une variable | `collect` |
| | Ordonner | `sort` |
| Calcul | Quotient de la division euclidienne | `quo` |
| | Reste de la division euclidienne | `rem` |
| | Test de divisibilité, division exacte | `divide`$^\star$ |
| | Pseudo-reste | `prem` |
| | Pseudo-reste creux | `sprem` |
| | Pgcd et pgcd étendu | `gcdex` |
| | Pgcd multivarié | `gcd`$^\star$ |
| | Plus petit commun multiple | `lcm`$^\star$ |
| | Normalisation des fractions | `normal`$^\star$ |
| | Numérateur | `numer`$^\star$ |
| | Dénominateur | `denom`$^\star$ |
| | Résultant | `resultant` |
| | Factorisation | `factor`$^\star$ |
| | Factorisation sans carrés | `convert(.,sqrfree)` |
| | Décomposition en éléments simples | `convert(.,parfrac)` |
| | Décomposition de polynômes | `compoly` |
| | Résolution exacte | `solve` |
| | Résolution numérique | `fsolve` |
| | Suites de STURM | `sturmseq` |
| | Localisation de racines réelles | `sturm` |
| | Manipulation de nombres algébriques | `evala` |

considérer comme des polynômes. Ce dernier point est particulièrement vrai en Maple, qui ne propose pas de type de donnée spécialisé pour les polynômes.

Ces opérations sont groupées en trois catégories : les opérations purement syntaxiques, qui extraient de l'information directement accessible sur le polynôme, les opérations de récriture, qui peuvent développer des produits pour réorganiser le polynôme, et les opérations impliquant un calcul, c'est-à-dire la division euclidienne et les opérations dérivées — pgcd, résultant et suites de STURM, normalisation de fractions rationnelles — ainsi que les commandes de factorisation et de résolution qui reposent sur des algorithmes

spécialisés. Dans ce tableau, sont marquées d'une étoile les opérations pour lesquelles on ne spécifie pas de variable privilégiée ; ces opérations traitent toutes les variables sur le même plan.

Nous allons maintenant reprendre un peu plus en détail les trois catégories de ce tableau. Notre but n'est pas de présenter la syntaxe de chacune de ces commandes, pour laquelle nous renvoyons le lecteur à l'aide en ligne (`help`), mais de préciser ou illustrer leur utilisation.

**1.1. Opérations purement syntaxiques.** Ces commandes ne font que parcourir l'expression pour en extraire l'information cherchée. Comme elles n'effectuent absolument aucun calcul, il faut que les polynômes soient développés par rapport à la variable d'intérêt, et que 0 ne soit pas caché dans un des coefficients utilisés.

EXEMPLE 1. Il ne faut donc pas faire :

```
degree((x-1)*x-x^2,x), degree((exp(a)*exp(-a)-1)*x+1,x);
```

qui renvoie les résultats faux :

$$2, \quad 1$$

Pour éviter la situation du premier calcul, on utilisera systématiquement la commande `collect` avant de calculer le degré ou la valuation. Quant au second résultat, il provient de ce que deux expressions transcendantes ($\exp(a)$ et $\exp(-a)$), mais algébriquement dépendantes, sont simultanément présentes dans l'expression, ce qu'il faut corriger dès détection (par exemple en utilisant `combine`).

**1.2. Récriture et simplification.** Un système ne peut pas décider *a priori* qu'une forme d'un polynôme est plus simple que les autres. Par exemple, $(1+x)^{100}$ peut être considérée comme une forme plus simple que la forme développée du même polynôme, alors que la forme factorisée de $1+x^{100}$ est plus compliquée que la forme développée si le critère retenu est la taille de l'expression. En outre, pour certains calculs comme la division euclidienne, la forme développée est plus commode, alors que pour d'autres calculs comme la recherche de pgcd, la forme factorisée permet des calculs plus rapides. En conséquence, c'est à l'utilisateur de décider de la forme la plus adaptée à ses besoins et Maple lui propose un certain nombre de commandes pour y parvenir.

1.2.1. *Développer*

La commande `expand`, présentée au chapitre I, développe tous les produits des polynômes, mais aussi toutes les sous-expressions pour lesquelles une des procédures `expand/.` s'applique (voir §II.3.3). Pour confiner la commande `expand` au développement de polynômes, on peut avoir recours à la commande `frontend` :

EXEMPLE 2.

```
P:=(sin(2*x)*t+1)^2:
expand(P), frontend(expand,[P]);
```
$$4t^2\sin(x)^2\cos(x)^2 + 4t\sin(x)\cos(x) + 1, \qquad \sin(2x)^2t^2 + 2\sin(2x)t + 1$$

Dans le cas d'expressions à plusieurs variables, ou dont les coefficients sont des fonctions de paramètres, `expand` présente l'inconvénient de trop développer. Cela prend du temps et produit des expressions très grosses où une partie de l'information peut être perdue. On aura souvent intérêt à utiliser la commande `collect` qui développe l'expression par rapport à une variable, sans modifier le reste. En outre, on peut donner un argument optionnel qui est une procédure à appliquer aux coefficients. Le plus souvent, on utilisera `normal`.

Exemple 3.

```
f:=(1+x*(1+y))^2/(1-y^2):
f=collect(f,x,normal);
```
$$\frac{(1+x(1+y))^2}{1-y^2} = \frac{(1+y)x^2}{1-y} + 2\frac{x}{1-y} + \frac{1}{1-y^2}$$

Pour développer des polynômes simultanément par rapport à plusieurs variables, on utilisera soit `collect(p,[x,y])` qui développe par rapport à $x$ puis les coefficients par rapport à $y$, soit `collect(p,[x,y],distributed)` qui développe "à plat"par rapport à $x$ et $y$.

```
p:=1+x*y+x*z+x*y*z:
collect(p,[x,y]), collect(p,[x,y],distributed);
```
$$((1+z)y+z)x+1, \quad 1+xz+(1+z)xy$$

1.2.2. *Ordonner les polynômes*

Du fait de la représentation des polynômes comme des sommes, Maple n'ordonne pas systématiquement les polynômes. Il faut explicitement le demander. Deux commandes permettent de réaliser cette opération. La commande `sort` *modifie* le polynôme afin de l'ordonner par puissances décroissantes de la ou des variables spécifiées (dans le cas de plusieurs variables on peut spécifier la relation d'ordre entre les monômes). La commande `series` ordonne un polynôme par puissances croissantes, il suffit de spécifier comme ordre du développement `infinity`. Dans ce dernier cas, l'objet renvoyé n'est plus un polynôme, mais un objet de type `series` (voir chap. II).

Cependant, pour l'écriture de programmes, il est généralement inutile d'ordonner les termes. On aura plutôt recours à la boucle `for ... in` et éventuellement aux commandes `degree` et `coeff`. La commande `sort` ne sert ainsi qu'à "faire joli".

**1.3. Calculs en une variable.** Les coefficients d'un polynôme sont *a priori* des expressions quelconques, mais les opérations de cette section nécessitent souvent que ces coefficients soient des polynômes ou des fractions rationnelles (en d'autres variables) dont les coefficients sont eux-mêmes rationnels. Par exemple le polynôme $p = x^2 + 2\sinh(y)x - 1$ est un multiple de $x - e^y$,

mais pour le déterminer, il faut récrire $\sinh(y)$ en termes de $z = e^y$ et faire les calculs sur des fractions rationnelles en $x$ et $z$. On peut espérer que les versions suivantes du système fonctionneront pour n'importe quelle extension algébrique, voire transcendante.

1.3.1. *Division euclidienne*

Étant donnés deux polynômes $A$ et $B$ en la variable $x$, il y a plusieurs façons en Maple d'obtenir les polynômes $Q$ et $R$ tels que $A = BQ + R$ et $\deg_x R < \deg_x B$. Les commandes principales sont `quo` et `rem`, qui renvoient respectivement le quotient $Q$ et le reste $R$ de la division.

EXEMPLE 4. Pour obtenir le reste de la division de $A = x^{10} + ux + 1$ par $B = 2x^3 + vx^2 - 1$ en la variable $x$, on fait :

```
A:=x^10+u*x+1: B:=2*x^3+v*x^2-1:
R:=rem(A,B,x);
```

$$R := \left(\frac{v^8}{256} - \frac{3v^5}{32} + \frac{3v^2}{8}\right)x^2 + \left(u + \frac{v^6}{128} + \frac{1}{8} - \frac{v^3}{8}\right)x + 1 - \frac{3v}{16} - \frac{v^7}{256} + \frac{5v^4}{64}$$

Si les coefficients de $A$ et $B$ sont rationnels, la fonction `divide` détermine si $B$ divise $A$. Cette fonction donne le quotient de manière plus rapide que `quo` lorsque la division est exacte. Enfin, les fonctions `prem` et `sprem` calculent un *pseudo-reste*, c'est-à-dire un polynôme $R'$ multiple par un scalaire du polynôme $R$. L'avantage de ces fonctions est qu'elles évitent les divisions de coefficients et donc sont souvent plus rapides sur de gros calculs.

EXEMPLE 5. On reprend les deux polynômes de l'exemple 4 :

```
prem(A,B,x);
```

$$256ux + 256 + 96v^2x^2 - 24v^5x^2 + 32x - 32xv^3 + 2xv^6 + v^8x^2 - 48v + 20v^4 - v^7$$

Ce polynôme vaut 256 fois celui obtenu par `rem` dans l'exemple 4, et on n'a pas eu à réduire toutes les fractions coefficients. La fonction `sprem` diffère de `prem` par le seul fait qu'elle cherche le plus petit facteur scalaire tel que $R'$ ne comporte pas de fraction.

1.3.2. *Pgcd*

Une fois que l'on sait effectuer une division euclidienne, on sait calculer le pgcd par l'algorithme d'EUCLIDE. Le pgcd $A \wedge B$ de deux polynômes $A$ et $B$ en une variable s'obtient par la commande `gcdex` qui prend en argument les polynômes et la variable.

EXEMPLE 6. Pour montrer qu'il n'existe pas de valeur de $(u, v)$ telle que les deux polynômes $A$ et $B$ de l'exemple 4 aient plus de deux racines communes, il suffit de faire :

```
gcdex(coeff(R,x,2),coeff(R,x,0),v);
```

1

Ceci montre que le reste de la division n'est jamais le polynôme nul, et par conséquent il y a au plus deux valeurs de $x$ pour lesquelles il s'annule.

Si l'on souhaite obtenir également les polynômes $U$ et $V$ tels que $AU + BV = A \wedge B$, on donne deux variables en arguments optionnels à `gcdex`. Une utilisation de ces polynômes est la manipulation symbolique de racines de polynômes sans en connaître l'expression formelle ou même la valeur numérique. Nous reviendrons en détail sur ce point en §1.3.8. Voici un premier cas particulier important.

Si $\alpha$ est racine d'un polynôme $H$, et $F$ une fraction rationnelle n'ayant pas de pôle en $\alpha$, on peut exprimer facilement $F(\alpha)$ comme un *polynôme* en $\alpha$ de degré inférieur à celui de $H$. Ceci fournit une *forme normale* qui permet de détecter syntaxiquement l'égalité de deux fractions rationnelles en $\alpha$.

Exemple 7. Par exemple si $\alpha$ est racine du polynôme $B = 2x^3 + vx^2 - 1$ des exemples précédents et $F = (x^2+1)/(x+1)$, on fait

```
F:=(x^2+1)/(x+1):
gcdex(denom(F),B,x,'U'):
subs(x=alpha,F=rem(U*numer(F),B,x));
```

$$\frac{\alpha^2+1}{\alpha+1} = \frac{v-1}{v-3} - \frac{(v-1)\alpha}{v-3} - 4\frac{\alpha^2}{v-3}$$

On obtient ainsi *le* polynôme de plus petit degré en $\alpha$ égal à $F(\alpha)$.

Il existe aussi en Maple une commande `gcd` qui donne le plus grand commun diviseur de deux polynômes multivariés à coefficients rationnels.

```
A:=x*(y-1)*(z-2): B:=x*(y-1)*(z-3):
gcd(A,B);
```

$$xy - x$$

1.3.3. *Résultants*

Étant donnés deux polynômes unitaires $A$ et $B$ à coefficients dans un anneau intègre, le résultant de $A$ et $B$ est un polynôme à coefficients dans le même anneau qui s'annule si et seulement si les deux polynômes ont une racine commune. Lorsque les polynômes ne sont pas unitaires, le résultant de $A$ et $B$ vaut $a^{\deg(B)}b^{\deg(A)}\text{Resultant}(A/a, B/b)$ où $a$ et $b$ sont les coefficients dominants de $A$ et $B$. Il n'y a toutefois plus équivalence entre l'annulation du résultant et l'existence d'une racine commune. Cette annulation peut en effet se produire également si l'un des deux polynômes devient nul ou si les coefficients de tête des deux polynômes s'annulent simultanément.

Exemple 8. La condition pour qu'un polynôme n'ait pas de racine multiple est qu'il n'ait pas de racine commune avec sa dérivée. Le résultant de $P$ et $P'$ s'appelle le *discriminant*, et on le trouve facilement :

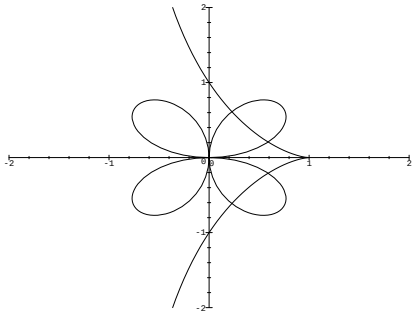


FIGURE 1

```
P:=a*x^2+b*x+c: Q:=x^3+p*x+q:
resultant(P,diff(P,x),x), resultant(Q,diff(Q,x),x);
```

$$4a^2c - b^2a, \qquad 4p^3 + 27q^2$$

Dans le premier exemple, on trouve un facteur $-a$ en plus du discriminant habituel $b^2 - 4ac$, qui reflète l'annulation simultanée des coefficients de tête des deux polynômes lorsque $a = 0$.

EXEMPLE 9. Si $\alpha$ est racine d'un polynôme $P(x)$, et $\beta$ est racine d'un polynôme $Q(x)$, on peut obtenir un polynôme qui s'annule en $\alpha + \beta$ par

```
resultant(P,subs(x=y-x,Q),x);
```

EXEMPLE 10. Les deux courbes définies par

$$(x^2 + y^2)^3 - 4x^2y^2 = 0 \qquad \text{et} \qquad y^2(1 + x) - (1 - x)^3 = 0$$

sont tracées en figure 1 p. 145. Le dessin montre que les courbes ont quatre points d'intersection. La détermination précise des coordonnées de ces points s'obtient en utilisant les résultants qui fournissent les équations aux abscisses et aux ordonnées des points d'intersection.

```
f:=(x^2+y^2)^3-4*x^2*y^2: g:=y^2*(1+x)-(1-x)^3:
resultant(f,g,y), resultant(f,g,x);
```

$$(-60x^6 - 4x^7 - 1 + 9x + 95x^3 - 35x^2 - 164x^4 + 152x^5)^2,$$
$$16y^{14} + 6032y^{12} - 1624y^{10} + 4192y^8 - 815y^6 - 301y^4 - 9y^2 + 1$$

Le premier polynôme est un carré alors que le second est bicarré. Cela s'explique par la symétrie de la figure par rapport à l'axe des abscisses. On note également que les degrés sont plus grands que le nombre de racines attendu. En les valeurs de $x$ ou de $y$ racines de ces résultants, ni $f$ ni $g$, ni leurs coefficients de tête ne s'annulent. Les racines des résultants qui ne sont pas des coordonnées des points d'intersections de la figure 1 correspondent donc à d'autres intersections de ces deux courbes, non plus dans $\mathbb{R}^2$, mais dans $\mathbb{C}^2$. Nous reviendrons sur cet exemple p. 148.

1.3.4. *Factorisation*

Bien que performantes, les méthodes de factorisation formelle de polynômes sont inévitablement très coûteuses en temps. Cela provient de l'existence de polynômes très simples avec un grand nombre de facteurs comme $x^{1000}-1$. Il est donc avisé d'éviter toute factorisation dans un programme, d'autant plus que l'on peut souvent s'en passer, car nombreuses sont les opérations qui donnent le même résultat symbolique pour toutes les racines d'un polynôme. Cependant, au cours d'une session interactive, lorsque l'on fait des expérimentations autour d'un problème, il peut se produire que la factorisation d'un polynôme apporte un éclairage intéressant. La commande à utiliser pour cela est `factor` qui factorise aussi bien des polynômes en une qu'en plusieurs variables. Lorsque `factor` renvoie le polynôme de départ, cela signifie que celui-ci est *irréductible* sur $\mathbb{Q}$.

Exemple 11. Dans l'exemple 10, les deux résultants restent inchangés par `factor`. Il s'agit donc des polynômes à coefficients dans $\mathbb{Q}$ de plus bas degré s'annulant aux coordonnées des intersections.

La commande `factor` s'applique aussi aux fractions rationnelles. Dans ce cas, numérateur et dénominateur sont factorisés.

1.3.5. *Factorisation sans carrés*

Cette opération réalisée uniquement à l'aide de pgcds n'est pas coûteuse. Elle permet de récrire un polynôme $P$ sous la forme $Q_1Q_2^2\cdots Q_p^p$, où les polynômes $Q_i$ sont premiers entre eux et n'ont pas de facteurs multiples.

Exemple 12. On prend $P = x^{14}+3x^{13}+3x^{12}+x^{11}-x^3-3x^2-3x-1$, la factorisation sans carrés s'obtient par :

```
convert(x^14+3*x^13+3*x^12+x^11-x^3-3*x^2-3*x-1,sqrfree,x);
```

$$(x^{11}-1)(x+1)^3$$

Outre son faible coût, cette opération présente deux avantages : elle permet d'extraire la partie sans carrés du polynôme (le produit des $Q_i$), c'est-à-dire un polynôme de degré minimal s'annulant aux mêmes points ; elle sépare également les racines par multiplicité, groupant ainsi celles qui se comportent de la même façon pour les opérations algébriques (par exemple la décomposition en éléments simples).

1.3.6. *Décomposition*

La décomposition de polynômes est une autre opération bien moins coûteuse que la factorisation et qui peut rendre service. Étant donné un polynôme $P$, il s'agit de déterminer l'existence de polynômes $Q$ et $R$ de degré au moins 2 tels que $P = Q(R)$. La commande Maple qui réalise ce calcul s'appelle `compoly`. Lorsque le système renvoie `FAIL`, cela signifie non pas qu'il n'a pas réussi à trouver de décomposition, mais qu'une telle décomposition n'existe pas.

1.3.7. *Résolution d'équations polynomiales*

La recherche de racines est un problème important en sciences appliquées. Lorsqu'elle est possible de manière exacte, elle sera obtenue par la commande `solve`.

EXEMPLE 13. On résout aisément l'équation du second degré :

```
solve(a*x^2+b*x+c,x);
```

$$\frac{-b+\sqrt{b^2-4ac}}{2a}, \qquad \frac{-b-\sqrt{b^2-4ac}}{2a}$$

On peut poursuivre ainsi jusqu'au degré 4. On sait qu'il n'existe pas de formule générale à l'aide de radicaux pour la résolution de polynômes de degré supérieur ou égal à 5. Il est parfois possible d'obtenir une telle formule par la théorie de GALOIS, mais Maple n'a pas de programme pour cela. Cependant, Maple essaye `factor` et `compoly`, ce qui lui permet parfois des résolutions spectaculaires.

EXEMPLE 14. Le polynôme

$$4x^8+48x^7+256x^6+792x^5+1590x^4+2196x^3+2104x^2+1290x+459,$$

est irréductible (d'après `factor`), et de degré 8. Cependant la résolution est aisée :

```
solve(4*x^8+48*x^7+256*x^6+792*x^5+1590*x^4+2196*x^3+
      2104*x^2+1290*x+459,x);
```

$$-3/2+\frac{\sqrt{-1+2\sqrt{-18+\sqrt{13}}}}{2}, \quad -3/2-\frac{\sqrt{-1+2\sqrt{-18+\sqrt{13}}}}{2},$$
$$-3/2+\frac{\sqrt{-1-2\sqrt{-18+\sqrt{13}}}}{2}, \quad -3/2-\frac{\sqrt{-1-2\sqrt{-18+\sqrt{13}}}}{2},$$
$$-3/2+\frac{\sqrt{-1+2\sqrt{-18-\sqrt{13}}}}{2}, \quad -3/2-\frac{\sqrt{-1+2\sqrt{-18-\sqrt{13}}}}{2},$$
$$-3/2+\frac{\sqrt{-1-2\sqrt{-18-\sqrt{13}}}}{2}, \quad -3/2-\frac{\sqrt{-1-2\sqrt{-18-\sqrt{13}}}}{2}$$

En règle générale, les solutions exactes de polynômes de degré supérieur à deux ne présentent pas d'intérêt. Il vaut mieux se demander ce que l'on cherche à faire avec ces solutions, et un calcul de résultant, une résolution numérique ou des manipulations de nombres algébriques que nous verrons en §1.3.8 permettent souvent d'aboutir au résultat.

La résolution numérique s'effectue avec la commande `fsolve`. Si l'on veut toutes les racines, il faut spécifier l'option `complex`, mais les algorithmes peuvent toujours être mis en défaut. Il se peut ainsi que `fsolve` renvoie des valeurs complexes alors que les racines sont réelles. On peut cependant compter le nombre de racines réelles de manière exacte (sans les calculer) par les commandes `sturmseq` et `sturm`. La première calcule la suite de STURM

associée au polynôme, qui est une suite de polynômes de degrés décroissants, et la seconde utilise cette suite pour calculer le nombre de racines dans n'importe quel intervalle réel.

EXEMPLE 15. On reprend l'exemple 10 de l'intersection de deux courbes.

```
readlib(sturm):
sturm(sturmseq(resultant(f,g,y),x),x,-infinity,infinity),
sturm(sturmseq(resultant(f,g,x),y),y,-infinity,infinity);
```

$$3, \quad 4$$

Le second résultat correspond bien à ce que l'on attend : on voit quatre intersections sur la figure, correspondant à quatre ordonnées distinctes. Le premier résultat est plus surprenant : on attendait 2, le nombre d'abscisses de points d'intersection. La troisième racine réelle, que l'on trouve numériquement par `fsolve`, correspond à deux intersections dont l'abscisse est réelle et l'ordonnée complexe :

```
fsolve(resultant(f,g,y),x);
```

$$-17.33341889, \quad 0.2277666333, \quad 0.5905668901,$$
$$-17.33341889, \quad 0.2277666333, \quad 0.5905668901$$

(Les racines sont renvoyées avec leur multiplicité ici égale à 2).

```
a:="[1]:
solve(subs(x=a,g),y);
```

$$-19.42346209i, \quad 19.42346209i$$

```
solve(subs(x=a,f),y);
```

$$19.42346207i, \quad -19.42346207i,$$
$$1.743900269 + 16.28118131i, \quad -1.743900269 - 16.28118131i,$$
$$1.743900269 - 16.28118131i, \quad -1.743900269 + 16.28118131i$$

Les deux premières racines sont bien communes aux deux polynômes, et correspondent à l'ordonnée des intersections. On remarque au passage que lorsqu'un coefficient du polynôme est numérique, `solve` se comporte comme `fsolve`.

### 1.3.8. *Nombres et fonctions algébriques*

Un *nombre algébrique* est un zéro d'un polynôme en une variable à coefficients rationnels. De même une *fonction algébrique* $f(x_1, \ldots, x_n)$ est une fonction annulant un polynôme en $f$ et $x_1, \ldots, x_n$ à coefficients rationnels. Alors qu'il est impossible de prouver des égalités entre nombres algébriques par des calculs numériques, il est possible de prouver de telles égalités lorsqu'elles ne font intervenir que des fractions rationnelles en des nombres algébriques, en utilisant des calculs similaires à celui de l'exemple 7. En outre, ces calculs mènent toujours à un résultat indépendant de la racine considérée.

En Maple, les nombres algébriques sont représentés à l'aide de la fonction `RootOf`, et peuvent être manipulés par la commande `evala`. L'opération de

normalisation mentionnée en §1.3.2 s'effectue alors directement, et le résultat de l'exemple 7 s'obtient simplement par :

```
alias(alpha=RootOf(2*x^3+v*x^2-1,x));
evala((alpha^2+1)/(alpha+1));
```

EXEMPLE 16. Le nombre d'or $\phi$ vaut $(\sqrt{5}+1)/2$. On remarque numériquement que

$$\frac{\phi^5+1}{\phi^2-1} \simeq 7,472\ 135\ 953, \qquad \frac{\phi^8}{\phi^3+1} - \frac{3}{2} \simeq 7,472\ 135\ 963.$$

Pour prouver l'égalité de ces deux nombres, il suffit de ramener chacun d'eux en forme normale.

```
alias(phi=RootOf(x^2-x-1)):
evala((phi^5+1)/(phi^2-1)),evala(phi^8/(phi^3+1)-3/2);
```

$$4\phi+1, \qquad 4\phi+1$$

Si $\alpha$ est un nombre algébrique racine d'un polynôme *irréductible*, on note habituellement $\mathbb{Q}(\alpha)$ le corps des fractions rationnelles en $\alpha$, qui est un espace vectoriel de dimension finie sur $\mathbb{Q}$, avec pour base $1, \alpha, \alpha^2, \ldots$ Cette structure particulièrement simple en fait un objet privilégié du calcul formel, auquel on aura souvent intérêt à se ramener. Lorsque plusieurs nombres algébriques sont en jeu, on peut toujours se ramener à des manipulations ne faisant intervenir qu'un nombre algébrique, en opérant comme dans l'exemple 9. Malheureusement, ces manipulations font intervenir des polynômes dont le degré peut devenir très grand, ce qui bloque parfois le calcul.

Il faut également savoir que l'on peut faire la plupart des opérations effectuées par `evala` sans que le polynôme définissant le nombre $\alpha$ soit nécessairement irréductible. Il faut simplement faire attention aux divisions qui peuvent nécessiter une discussion. Malheureusement, Maple est souvent réticent à ce genre de calcul et renvoie des messages d'erreur inopportuns :

```
evala(subs(alpha=RootOf(x^2-4),1/alpha));
Error, (in evala) reducible RootOf detected.
   Substitutions are, {RootOf(_Z^2-4) = 2, RootOf(_Z^2-4) = -2}
```

au lieu de $\alpha/4$ qui est le résultat correct.

Il est également important de se ramener à des nombres algébriques pour *prouver* des inégalités par le biais de suites de STURM. En effet, le calcul numérique est toujours entaché d'erreurs d'arrondi. Si l'on souhaite prouver que $\phi = (\sqrt{5}+1)/2 < 75025/46368$, on ne peut pas se contenter d'une évaluation à 10 décimales :

```
(sqrt(5.)+1)/2-75025/46368;
```

$$0$$

On peut bien sûr augmenter la précision, mais il est impossible de savoir *a priori* quelle est la précision nécessaire. On procèdera donc plutôt ainsi :

```
sturm(sturmseq(x^2-x-1,x),x,0,75025/46368);
```

1

Cette réponse prouve que le polynôme $x^2-x-1$ a une racine dans l'intervalle $]0, 75025/46368]$. Comme d'autre part le produit des racines vaut -1, $\phi$ est bien la seule racine positive et on vient donc de prouver qu'il est inférieur à 75025/46368. On sait par ailleurs qu'il n'y a pas égalité, car `evala` aurait alors renvoyé des formes normales rationnelles dans l'exemple 16.

1.3.9. *Fonctions symétriques des racines*

La fonction `sum` permet de calculer directement certaines fonctions symétriques des racines.

EXEMPLE 17. Si $x_1, x_2, x_3, x_4$ sont les racines de $x^4+px+q$, la somme des $x_i^{10}$ est obtenue par

```
sum(x^10,x=RootOf(x^4+p*x+q,x));
```

$$-10qp^2$$

Ce procédé s'applique à toutes les expressions $\sum_i F(x_i)$ où $F(x)$ est une fraction rationnelle en $x$.

### 1.4. Exercices.

1. Lorsque le degré d'un polynôme est une variable, les fonctions présentées dans ce chapitre ne fonctionnent plus. Il faut alors assister les systèmes de calcul formel. À titre d'exemple, calculer le reste de la division euclidienne de $x^n$ par $x^2-2\cos(\phi)x+1$ ($n \in \mathbb{N}$).
2. Calculer la décomposition en éléments simples sur $\mathbb{R}$ de
$$\frac{7}{(x+1)^7-x^7-1}.$$
3. Calculer un polynôme dont les racines sont les numérateurs de la décomposition en éléments simples sur $\mathbb{C}$ de $(x^9+1)/(x^{10}-1)$, sans calculer cette décomposition.
4. À l'aide des fonctions `evalc`, `expand`, et `simplify(.,trig)`, montrer que l'on a la décomposition en éléments simples suivante :
$$\frac{1}{x^{2n}-1} = \frac{1}{2n}\left(\frac{1}{x-1}-\frac{1}{x+1}\right)+\frac{1}{n}\sum_{k=1}^{n-1}\frac{\cos(k\pi/n)x-1}{x^2-2\cos(k\pi/n)x+1}.$$
Indication : les pôles sont en $r_k=\exp(ik\pi/n)$, et le résidu y vaut
$$\frac{1}{2nr_k^{2n-1}}.$$

## 2. Polynômes et systèmes multivariés

L'étude des polynômes et de leurs racines est un domaine très riche, qui fait le lien entre la géométrie et l'algèbre. Les relations géométriques entre des points du plan ou de l'espace peuvent souvent se traduire en systèmes d'équations polynomiales satisfaites par leurs coordonnées, systèmes dont on

veut déduire des informations comme la cohérence (existe-t-il une solution ?), la dimension de l'espace des solutions (les solutions sont-elles en nombre fini ou infini ?), la simplification (une fois toutes les coordonnées contraintes par le système, combien vaut telle quantité ?), l'élimination (peut-on trouver une relation entre les coordonnées de tel point qui ne fasse plus apparaître tel paramètre ?),...

Les *bases de* GRÖBNER (on dit aussi *bases standard*) sont l'outil fondamental pour toutes ces questions. Sans entrer dans la théorie, nous allons montrer comment utiliser cet outil pour résoudre un certain nombre de problèmes liés aux systèmes de polynômes et à leur utilisation géométrique. Il faut cependant savoir que les calculs de bases de GRÖBNER sont extrêmement coûteux, de par la nature même du problème. On ne peut donc espérer s'attaquer de cette façon à des systèmes avec beaucoup de variables ou contenant des équations de degré élevé.

Pour préciser le vocabulaire, dans la suite de ce chapitre on dira que l'on peut *déduire* un polynôme d'un système de polynômes lorsqu'il est possible de l'obtenir par combinaison linéaire à coefficients polynomiaux des polynômes du système.

## 2.1. Bases de Gröbner.

### 2.1.1. *Introduction*

Nous avons vu dans la section précédente le rôle important que jouait la division euclidienne pour fournir des *formes normales* : la forme normale d'un polynôme en un nombre algébrique est donnée par le reste de la division euclidienne du polynôme par celui qui définit le nombre. L'intérêt des bases de GRÖBNER est de fournir des formes normales lorsque l'on part des solutions d'un système de polynômes en plusieurs variables au lieu de partir des solutions d'un seul polynôme en une seule variable.

Partant d'un système de polynômes comme

$$\left\{x^2+2y^2-yz-1, x+z^3-1, x^2+y^2+z^2+1\right\},$$

on peut trouver (*déduire* au sens introduit plus haut) d'autres polynômes plus simples qui s'annulent aux mêmes points que ces polynômes. Par exemple, en multipliant le 3$^{\text{e}}$ polynôme par $y$ et en y ajoutant le 1$^{\text{er}}$ multiplié par $z$, on en fait décroître le degré en $z$. Cette opération peut se répéter avec d'autres monômes d'autres polynômes. On obtient ainsi un système de polynômes plus simples, qui s'annulent aux mêmes points que les polynômes de départ. Chaque étape ressemble à une étape de la division euclidienne, où on annule un par un les monômes de plus haut degré du polynôme à diviser, et ce degré décroît à chaque étape. La différence avec la division euclidienne réside en un choix à réaliser au départ : en plusieurs variables, il y a généralement plusieurs monômes de chaque degré, et il faut choisir dans quel ordre on décide de les annuler. Plusieurs ordres sur les monômes sont utilisés en pratique. Si les

monômes à ordonner sont $A = x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$ et $B = x_1^{b_1} x_2^{b_2} \cdots x_n^{b_n}$, avec $\alpha = a_1 + \cdots + a_n$ et $\beta = b_1 + \cdots + b_n$, alors les ordres principaux sont :

- l'ordre *lexicographique*, où $A > B$ si et seulement si la première coordonnée non nulle est positive dans le vecteur $(a_1 - b_1, \ldots, a_n - b_n)$ ;
- l'ordre du *degré lexicographique*, où $A > B$ si et seulement si $\alpha > \beta$, ou $\alpha = \beta$ et $A > B$ pour l'ordre lexicographique ;
- l'ordre du *degré lexicographique inverse*, où $A > B$ si et seulement si $\alpha > \beta$, ou $\alpha = \beta$ et dans le vecteur $(a_1 - b_1, \ldots, a_n - b_n)$, la dernière coordonnée non nulle est négative.

Maple connaît le premier et le troisième, sous les noms `plex` et `tdeg` respectivement. On spécifie l'ordre sur les variables (laquelle est $x_1$, $x_2$,...) en donnant une liste en argument.

Sans être plus précis, disons qu'une base de GRÖBNER est un système déduit du système de départ, où on ne peut plus effectuer aucune réduction compte tenu de l'ordre choisi. Étant donné un système de polynômes et un ordre sur les monômes, il y a unicité de la base. Cependant, la base dépend de l'ordre choisi.

EXEMPLE 18. Avec l'exemple précédent, nous obtenons ainsi trois bases :

```
sys:={x^2+y^2+z^2+1,x^2+2*y^2-y*z-1,x+z^3-1}:
grobner[gbasis](sys,[x,y,z],plex);
```

$$[x + z^3 - 1, \quad -5z^7 + 10z^4 - 5z^3 - 10z + 4z^8 - 8z^5 + 8z^2 - z^{11} + 4y, \\ 12z^6 - 16z^3 + 18z^2 + 16 + 5z^8 - 10z^5 + 5z^4 - 4z^9 + z^{12}]$$

```
grobner[gbasis](sys,[x,y,z],tdeg);
```

$$[x + z^3 - 1, \quad y^2 - z^2 - 2 - yz, \quad x^2 + yz + 3 + 2z^2]$$

```
grobner[gbasis](sys,[y,z,x],tdeg);
```

$$[x + z^3 - 1, \quad x^2z^2 + 3z^2 - 2zx + 2z - xy + y, \\ -3zx^2 - 7z + yx^2 + 3y + 5x - 5, \quad x^2 + y^2 + z^2 + 1, \\ x^4 - 2z^2 + 6x^2 + 9 + 5zx - 5z + 5xy - 5y, \quad x^2 + yz + 3 + 2z^2]$$

Nous insistons une fois encore sur l'énorme coût des calculs de bases de GRÖBNER. Ce coût dépend fortement de l'ordre choisi. Dans certains calculs, on a besoin d'un ordre particulier pour le résultat à trouver, mais lorsque l'on a le choix, alors il vaut mieux utiliser l'ordre `tdeg` qui donne en général une base plus rapidement.

### 2.1.2. *Résolution d'équations*

Un certain nombre d'informations se lisent directement sur la base de GRÖBNER. La plus simple est la cohérence du système : un système de polynômes n'a pas de solution (sur $\mathbb{C}$) si et seulement si la base est réduite au polynôme 1.

EXEMPLE 19. On considère le système précédent auquel on rajoute l'équation $x^3 + 2xy = 2$. Il n'y a alors plus de solution :

```
sys2:=sys union {x^3+2*x*y-2}:
grobner[gbasis](sys2,[x,y,z],tdeg);
```

[1]

La commande `solvable` du *package* `grobner` permet de répondre directement à la question de la cohérence.

Il est également possible de déterminer si un système a un nombre fini de solutions (toujours sur $\mathbb{C}$). Cela se produit si et seulement si chacune des variables apparaît seule, éventuellement élevée à une puissance et multipliée par une constante, comme monôme de tête (pour l'ordre choisi) d'un des éléments de la base. Cette propriété est vraie pour tout ordre, et on a à nouveau intérêt à calculer la base pour l'ordre `tdeg`.

EXEMPLE 20. Dans l'exemple 18, le système n'a qu'un nombre fini de solutions : sur la première base donnée en exemple, les monômes de tête sont $x$, $4y$ et $z^{12}$, sur la deuxième $z^3$, $y^2$ et $x^2$, et sur la troisième $z^3$, $x^2z^2$, $yx^2$, $y^2$, $x^4$ et $yz$.

La commande `finite` du *package* `grobner` permet de répondre directement à la question du nombre fini de solutions.

Pour la résolution, on peut utiliser la base calculée par l'ordre lexicographique, qui donne une *forme triangulaire* au système : le nombre de variables intervenant dans les équations va en décroissant. Cette façon de faire n'est pas la plus efficace dans le cas général, et on aura intérêt à utiliser directement `solve`, qui effectue un calcul basé sur des bases de GRÖBNER en tenant compte de factorisations intermédiaires.

EXEMPLE 21. Sur notre exemple précédent, on obtient les solutions :

```
solve(sys,{x,y,z});
```

$$\Big\{z = \%_1, \quad x = -{\%_1}^3 + 1,$$
$$y = \frac{5}{2}\%_1 - {\%_1}^8 + 2{\%_1}^5 - 2{\%_1}^2 + \frac{5}{4}{\%_1}^3 + \frac{5}{4}{\%_1}^7 + \frac{{\%_1}^{11}}{4} - \frac{5}{2}{\%_1}^4\Big\}$$
$$\%_1 = \mathrm{RootOf}\big(5\_Z^8 - 10\_Z^5 + 18\_Z^2 + 5\_Z^4 + \_Z^{12} - 4\_Z^9 + 12\_Z^6 - 16\_Z^3 + 16\big)$$

On a donc 12 solutions : une pour chacune des valeurs de $z$. Parmi celles-ci aucune n'est réelle, ce que l'on peut montrer à l'aide des suites de STURM.

Comme dans le cas des polynômes à une variable, la résolution explicite n'est souvent utile qu'en dernière phase d'un calcul, car beaucoup d'informations peuvent s'obtenir plus facilement à partir du système lui-même qu'à partir de l'ensemble de solutions.

2.1.3. *Forme normale des polynômes*

De même que la division euclidienne fournit une forme normale pour les polynômes et les fractions rationnelles en des nombres algébriques, on obtient grâce aux bases de GRÖBNER une forme normale pour les polynômes en des racines d'un système de polynômes. Cela permet de prouver des théorèmes de nature géométrique de la forme "on construit un point ayant telle et telle propriété, alors il a également telle autre propriété".

Deux façons de procéder existent en Maple : soit on calcule d'abord une base et on applique la commande `normalf` du *package* `grobner`, qui réduit le polynôme par rapport à cette base ; soit on utilise la commande `simplify`, en lui donnant en dernier argument l'ensemble de polynômes.

EXEMPLE 22. Il n'est pas évident *a priori* que l'on puisse déduire le polynôme

$$-x^2y + 2xy^2 + 2y^3 + xz^2 - 2y^2 + yz - 2z^2 + y - z - 1$$

du système $\{x^2y + 2y^2 + z + 1, xy^2 - z^2\}$. Voici comment on peut *prouver* que c'est le cas :

```
p1:=x^2*y+2*y^2+z+1: p2:=x*y^2-z^2:
q:=-x^2*y+2*x*y^2+2*y^3+x*z^2-2*y^2+y*z-2*z^2+y-z-1:
g:=grobner[gbasis]({p1,p2},[x,y,z],tdeg):
grobner[normalf](q,g,[x,y,z],tdeg);
```

$$0$$

On obtient le même résultat plus simplement par

```
simplify(q,{p1,p2});
```

qui appelle d'ailleurs `grobner[gbasis]` et `grobner[normalf]` (avec l'ordre lexicographique).

On peut en plus se demander quelle est la combinaison de $p_1$ et $p_2$ qui donne $q$. Nous reviendrons sur cette question en §2.2.1.

EXEMPLE 23. Voici un exemple de nature géométrique représenté en figure 2. On considère un cercle de rayon $R$ centré à l'origine, et deux points $A$ et $B$ sur l'axe des $x$, symétriques par rapport à l'origine. Étant donné un point $P$ du cercle, les droites $PA$ et $PB$ coupent le cercle respectivement aux points $C$ et $D$. La droite $CD$ coupe l'axe des abscisses en un point $E$. Soit $Q$ le symétrique de $P$ par rapport à l'origine, il s'agit de montrer que la tangente en $Q$ au cercle passe par $E$.

Pour résoudre ce problème, on se contente de le mettre en équations, après avoir pris $R = 1$, ce qui ne restreint pas la généralité mais permet de réduire le nombre de variables.

Les symétries et la position des points sur l'axe des $x$ donnent

$$x_B = -x_A, \quad x_Q = -x_P, \quad y_Q = -y_P, \quad y_A = y_B = y_E = 0\ ;$$

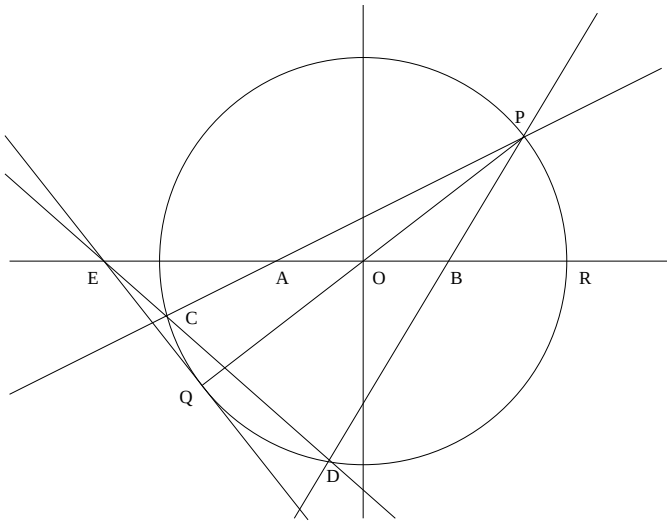



FIGURE 2

les alignements de points nous donnent

$$(x_C - x_P)(y_A - y_P) = (x_A - x_P)(y_C - y_P),$$
$$(x_D - x_P)(y_B - y_P) = (x_B - x_P)(y_D - y_P),$$
$$(x_E - x_C)(y_C - y_D) = (x_C - x_D)(y_E - y_C)\ ;$$

et les trois points sur le cercle nous donnent les équations :

$$x_P^2 + y_P^2 = 1, \quad x_C^2 + y_C^2 = 1, \quad x_D^2 + y_D^2 = 1.$$

Il faut encore exprimer que $C \neq P$ et $D \neq P$, et que les points $C$ et $D$ ne doivent pas avoir la même ordonnée, puisqu'alors la droite $CD$ n'aurait pas d'intersection avec l'axe des abscisses. Pour cela on rajoute un paramètre $t$ et on prend l'équation

$$1 - t(x_P - x_C)(x_P - x_D)(y_C - y_D) = 0 \tag{1}$$

qui ne peut avoir de solution que lorsque les inéquations sont vérifiées. On veut prouver que la droite $QE$ est perpendiculaire au rayon $OQ$, c'est-à-dire

$$x_Q(x_Q - x_E) + y_Q(y_Q - y_E) = 0. \tag{2}$$

Il suffit de poser la question :

```
sys:={xb+xa,xq+xp,yp+yq,ya,yb,ye,xp^2+yp^2-1,
xc^2+yc^2-1,(xc-xp)*(ya-yp)-(xa-xp)*(yc-yp),
xd^2+yd^2-1,(xd-xp)*(yb-yp)-(xb-xp)*(yd-yp),
(xe-xc)*(yc-yd)-(xc-xd)*(ye-yc),1-t*(xp-xc)*(xp-xd)*(yc-yd)}:
vars:=[ya,yb,ye,xa,xb,xp,yp,xq,yq,xc,yc,xd,yd,xe,t]:
g:=grobner[gbasis](sys,vars,tdeg):
grobner[normalf](xq*(xq-xe)+yq*(yq-ye),g,vars,tdeg);
```

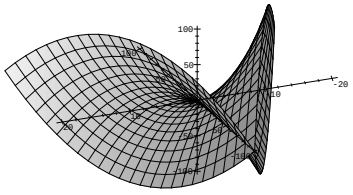


FIGURE 3

0

Ce qui prouve que l'équation (2) découle des hypothèses et donc la propriété est démontrée.

## 2.2. Applications.

2.2.1. *Élimination*

Comme nous l'avons déjà remarqué plus haut, les bases de GRÖBNER pour l'ordre lexicographique fournissent une forme triangulaire du système. À la manière des résultants dans le cas où l'on avait deux polynômes, ces bases fournissent donc des polynômes où l'on a *éliminé* certaines des variables.

EXEMPLE 24. La surface représentée en figure 3 a été tracée grâce à la représentation paramétrique

$$x = uv, \quad y = u + v, \quad z = u^2 - v^2$$

par la commande

```
plot3d([u*v,u+v,u^2-v^2],u=-10..10,v=-10..10);
```

On peut obtenir une relation entre $x$, $y$ et $z$ ainsi :

```
sys:={x-u*v,y-(u+v),z-(u^2-v^2)}:
grobner[gbasis](sys,[u,v,x,y,z],plex);
```

$$[-y + u + v, \quad 2v^2 + 2x + z - y^2, \quad z - y^2 + 2vy,$$
$$- yz + y^3 - 4yx + 2vz, \quad z^2 - y^4 + 4y^2x]$$

Le dernier polynôme de cette base (et lui seul) ne fait apparaître ni $u$, ni $v$, et donne donc l'équation cherchée.

Plus généralement, tous les polynômes que l'on peut déduire d'un système et dans lesquels certaines des variables ont été éliminées peuvent également être déduits de la base calculée pour l'ordre lexicographique. Pour cela il suffit de mettre les variables à éliminer en premier. Quand, comme dans notre exemple, il n'y a qu'un polynôme dans la base pour lequel les variables ont été éliminées, cela signifie que seuls les multiples de ce polynôme répondent à la question.

Une autre utilisation de l'élimination est la recherche d'une relation de liaison entre des polynômes.

EXEMPLE 25. Les polynômes $p_1$, $p_2$ et $q$ introduits dans l'exemple 22 sont liés. Pour trouver une relation entre eux, on ajoute une variable que l'on élimine aussitôt :

```
grobner[gbasis]({u-t*p1,v-t*p2,w-t*q},[t,w,u,v,x,y,z],plex);
```

$$[-u + tx^2y + 2ty^2 + tz + t, \quad -v + txy^2 - tz^2,$$
$$tz^2x + 2ty^3 + tzy + ty - yu + xv,$$
$$2y^5t + y^3tz + ty^3 + z^4t - uy^3 + z^2v + xy^2v,$$
$$-2v + u + w - yu + xv, \quad -v - zv - 2y^2v - z^2u + xy^2u - x^2yv]$$

Le premier des polynômes où n'apparaît pas $t$ donne la relation de liaison cherchée :

$$q = (y-1)p_1 + (2-x)p_2.$$

L'ordre que nous avons choisi sur les variables permet d'avoir une base assez petite, mais tout autre ordre convient, à condition que $t$ apparaisse en premier.

REMARQUE : lorsqu'on cherche une relation de la forme $q = ap_1 + bp_2$ entre trois polynômes $q, p_1, p_2$, une autre méthode consiste à employer une forme spéciale de `gcdex` :

```
gcdex(p1,p2,q,x,'a','b'): a,b;
```

$$y - 1, \quad 2 - x$$

Mais contrairement à l'approche utilisant les bases de GRÖBNER, cette méthode ne fonctionne plus lorsqu'on ajoute d'autres polynômes $p_i$.

Il est un peu dommage de calculer toute la base de GRÖBNER pour l'ordre lexicographique lorsque l'on ne veut, comme dans l'exemple ci-dessus, éliminer qu'une variable. C'est d'autant plus gênant que les calculs de base pour l'ordre lexicographique sont très coûteux, et on court le risque de ne pas parvenir au résultat en un temps ou un espace mémoire raisonnable parce que l'on calcule beaucoup plus que nécessaire. Il est possible de faire moins de travail en utilisant des ordres sur les monômes spécialisés pour l'élimination. Malheureusement, ces ordres ne sont pas disponibles en Maple, et l'utilisateur devra alors se tourner vers un système spécialisé sur les calculs de bases de GRÖBNER, comme les logiciels Macaulay ou Gb (voir Ann. C).

Nous n'insistons pas ici sur les précautions à prendre lorsque l'on utilise les systèmes triangulaires obtenus pour l'ordre lexicographique pour "remonter" de la dernière équation à la première. Il faut en effet garder à l'esprit que toutes les solutions de la dernière équation ne correspondent pas nécessairement à des solutions du système au complet. Cependant le fait que les monômes de tête des équations ne s'annulent pas en la solution garantit cette correspondance et cette condition suffisante est d'application fréquente. C'est en particulier toujours le cas si les monômes de tête de la base sont le produit d'une constante par une puissance d'une variable.

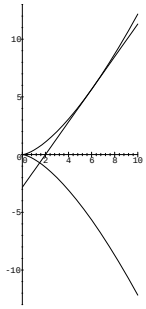



FIGURE 4

EXEMPLE 26. On cherche une droite qui soit à la fois tangente et normale à la courbe suivante, représentée en fig. 4 :

$$x = 3t^2, \qquad y = 2t^3.$$

En un point de paramètre $t_0$, la pente de la tangente est $(dy/dt)/(dx/dt) = t_0$. La tangente a donc pour équation paramétrique

$$x = x_0 + u, \qquad y = y_0 + t_0 u.$$

Pour qu'elle soit normale à la courbe, il faut qu'elle atteigne la courbe en un point de paramètre $t_1$ où le vecteur tangent est orthogonal à $(1, t_0)$. Autrement dit $1 + t_0 t_1 = 0$. On calcule donc

```
sys:={x0-3*t0^2,y0-2*t0^3,x1-(x0+u),y1-(y0+t0*u),x1-3*t1^2,
      y1-2*t1^3,1+t0*t1}:
grobner[gbasis](sys,[x0,y0,x1,y1,u,t1,t0],plex);
```

$$[x_0 - 3{t_0}^2, \quad y_0 - 2{t_0}^3, \quad 2x_1 - 3{t_0}^4 + 9, \quad 2y_1 - 3{t_0}^5 + 2{t_0}^3 + 9t_0,$$
$$2u - 3{t_0}^4 + 6{t_0}^2 + 9, \quad 2t_1 + {t_0}^5 - 3t_0, \quad -2 + {t_0}^6 - 3{t_0}^2]$$

La dernière équation donne les valeurs du paramètre $t_0$. La condition suffisante ci-dessus s'applique : comme tous les polynômes ont pour monôme dominant une puissance d'une variable multipliée par une constante, chaque solution de la dernière équation correspond à une solution du système. Pour y voir plus clair on factorise le dernier polynôme :

```
factor("[nops(")]);
```

$$(t_0^2 - 2)(t_0^2 + 1)^2$$

Il n'y a donc que deux racines réelles : $\pm\sqrt{2}$, et on en déduit les équations des deux droites symétriques solutions du problème :

$$x = 6 + u, \qquad y = \pm\sqrt{2}(4 + u).$$

La figure 4 a été obtenue par la commande

```
plot({[3*t^2,2*t^3,t=-2..2],[6+u,sqrt(2)*(4+u),u=-6..4]},0..10,
    scaling=CONSTRAINED);
```

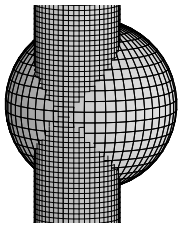


FIGURE 5

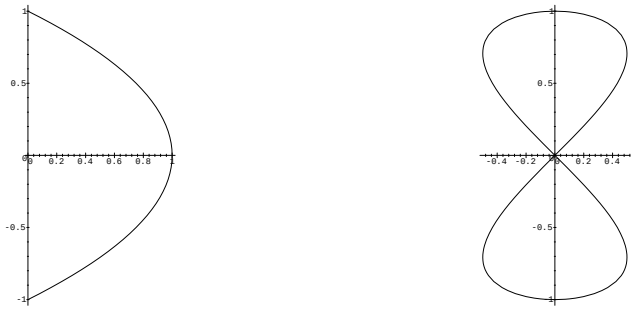



FIGURE 6

Géométriquement, l'élimination correspond à une projection. On peut donc calculer assez facilement l'équation de la projection d'une courbe de l'espace sur un plan de coordonnée.

EXEMPLE 27. La courbe de VIVIANI est l'intersection (fig. 5 p. 159) de la sphère $x^2+y^2+z^2=1$ et du cylindre $x^2+y^2-x=0$. Voici comment on calcule les équations de ses projections sur $Oxz$ et sur $Oyz$ :

```
sys:={x^2+y^2+z^2-1,x^2+y^2-x}:
grobner[gbasis](sys,[y,x,z],plex),
grobner[gbasis](sys,[x,y,z],plex);
```

$$[y^2+z^4-z^2, z^2+x-1], \quad [z^2+x-1, y^2+z^4-z^2]$$

Dans chaque cas, la seconde équation est celle que l'on cherchait. Ces courbes sont tracées en figure 6. La première est une portion de parabole, et la seconde s'appelle la *lemniscate de* GERONO.

### 2.2.2. *Calcul d'enveloppes*

Les enveloppes de familles paramétrées de courbes ou de surfaces algébriques sont elles-mêmes algébriques. Pour calculer l'enveloppe d'une famille de courbes d'équation $f(x,y,\lambda)=0$, on utilise le fait que les coordonnées des points de l'enveloppe satisfont le système :

$$\begin{cases} f(x,y,\lambda)=0, \\ f'_\lambda(x,y,\lambda)=0. \end{cases}$$

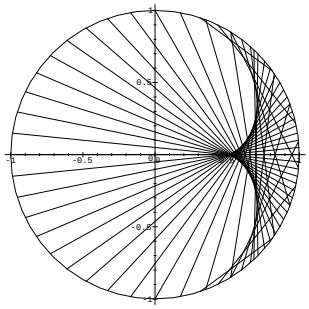


FIGURE 7 Lumière dans une tasse à café.

Il ne reste plus alors qu'à éliminer $\lambda$ de ces deux équations pour obtenir l'équation de l'enveloppe.

EXEMPLE 28. Lorsque la lumière arrive à l'intérieur d'une tasse à café à bord circulaire, elle est réfléchie par le cercle. Les rayons lumineux sont représentés en figure 7. On voit distinctement leur enveloppe dans la tasse "réelle" comme une courbe lumineuse. Cette courbe s'appelle une *néphroïde* ou une *épicycloïde de* HUYGENS, du nom de son découvreur. On se propose de déterminer l'équation de cette courbe sachant que les rayons de lumière venant du soleil peuvent être considérés comme parallèles.

On prend le cercle de rayon 1 centré sur l'origine. On considère un point de coordonnées $(x, y)$ du cercle. Si $\theta = \arctan(y/x)$, un rayon horizontal réfléchi en $(x, y)$ a un vecteur directeur de coordonnées

```
normal(expand(subs(th=arctan(y/x),[cos(2*th),sin(2*th)])));
```

$$\frac{x^2 - y^2}{x^2 + y^2}, \qquad \frac{2xy}{x^2 + y^2}.$$

Donc les droites qui nous intéressent ont pour équation

$$2xy(X - x) - (x^2 - y^2)(Y - y) = 0.$$

Dans notre exemple, on prend $\lambda = y$, et voici l'équation de la courbe :

```
cercle:=x(y)^2+y^2-1:
dx:=solve(diff(cercle,y),diff(x(y),y)):
eq:=2*x(y)*y*(X-x(y))-(x(y)^2-y^2)*(Y-y):
eq2:=numer(subs(diff(x(y),y)=dx,diff(eq,y))):
sys:=subs(x(y)=x,{cercle,eq,eq2}):
grobner[gbasis](sys,[x,y,X,Y],plex);
```

$$
\begin{aligned}
&[3x^2 + 4X^2 - 4 + 4Y^2, \quad xy + 2xY - 2yX,\\
&9Xx + 16X^4 + 32Y^2X^2 - 14X^2 + 16Y^4 - 2 - 14Y^2,\\
&72xY^2 + 9x - 128X^5 - 256X^3Y^2 + 112X^3 - 128Y^4X - 38X + 112Y^2X,\\
&3y^2 - 4X^2 + 1 - 4Y^2,\\
&36yX^2 - 9y + 64YX^4 - 23Y - 32X^2Y - 32Y^3 + 128Y^3X^2 + 64Y^5,\\
&9yY - 32Y^2X^2 + 8X^2 - 16X^4 - 16Y^4 - 1 + 8Y^2,\\
&-1 - 15Y^2 + 12X^2 - 96Y^2X^2 - 48Y^4 - 48X^4\\
&\qquad + 64Y^6 + 192Y^4X^2 + 192X^4Y^2 + 64X^6]
\end{aligned}
$$

Le dernier polynôme, où n'interviennent que $X$ et $Y$, donne l'équation cherchée. On peut alors tracer la courbe par `implicitplot` du *package* `plots`, mais on a plutôt intérêt pour obtenir un beau tracé à la paramétrer en prenant son intersection avec les droites $Y = tX$.

### 2.2.3. *Précautions géométriques*

Les calculs de bases de GRÖBNER ont bien entendu un lien avec la géométrie. Comme on l'a vu dans l'exemple 23, on définit des droites, des cercles, des ellipses ou de nombreuses autres courbes ou surfaces par des équations polynomiales, et diverses questions relatives à leurs intersections peuvent être résolues à l'aide de bases de GRÖBNER.

Il faut néanmoins faire une distinction entre les besoins que l'on a lors du calcul de forme normale de polynômes par exemple, et les besoins que l'on a lors de calculs géométriques. Nous avons dit que les bases de GRÖBNER permettaient de déterminer si un polynôme pouvait se déduire d'un autre système de polynômes, en définissant la déduction comme une combinaison linéaire à coefficient polynomiaux. La déduction géométrique est d'une autre nature : l'objet d'étude n'est plus le polynôme, mais l'ensemble de ses zéros — *la variété* qu'il définit. Les déductions que l'on cherche à faire sont du type : "est-ce que tel polynôme est nul en l'ensemble des zéros du système ?". Or un polynôme peut être nul en l'ensemble des zéros d'un système sans que l'on puisse pour autant le déduire au sens précédent du système. Par exemple, si l'on part du système réduit à un polynôme $\{x^2\}$, on ne peut pas déduire au sens précédent que $x = 0$.

Pour les déductions géométriques, il ne suffit donc pas de tester si le polynôme se réduit à 0 par une base de GRÖBNER, il faut en réalité tester si une puissance du polynôme se réduit à 0. Malheureusement, on n'a pas connaissance *a priori* de la puissance nécessaire.

On peut quand même faire le calcul à l'aide de bases de GRÖBNER, au prix de l'ajout d'une variable. Si $\mathcal{S} = \{f_1, \ldots, f_k\}$ est le système, et $g$ le polynôme dont on cherche à prouver qu'il s'annule sur la variété définie par $\mathcal{S}$, on calcule la base de GRÖBNER du système $\mathcal{S} \cup \{1 - tg\}$, où $t$ est une variable annexe. Si $g$ s'annule en toutes les racines de $\mathcal{S}$, alors le système n'a pas de solution et la base doit être réduite à $\{1\}$.

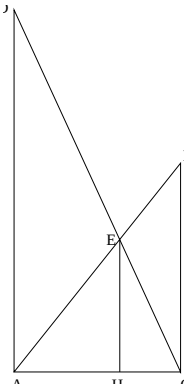



FIGURE 8

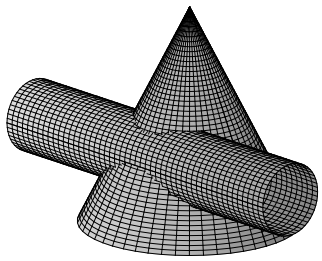


FIGURE 9

Cependant les calculs deviennent bien plus coûteux, et en général pour les problèmes "bien posés" on n'a pas à faire intervenir cette variable supplémentaire. La difficulté en pratique vient plutôt de ce qu'il faut penser à rajouter un certain nombre de conditions pour éviter les cas de dégénérescence, comme l'équation (1) de l'exemple 23.

### 2.3. Exercices.

1. Déterminer $\lambda$ pour que les deux droites
$$x-1=\frac{y-2}{2}=z+\lambda \quad \text{et} \quad \begin{cases} 2x+y+2z=\lambda \\ x+y+z=\frac{1}{3} \end{cases}$$
se rencontrent.
2. Sur la figure 8, on connaît les longueurs suivantes :
$$AB=2, \qquad DC=3, \qquad EH=1.$$
Que vaut $AC$ ? [Indication : il faut rajouter une condition qui est implicite sur le dessin pour qu'il n'y ait que deux solutions réelles. On pourra utiliser la fonction `solve` pour voir si l'on a bien spécifié le système. Numériquement on doit trouver deux valeurs proches de 1,23 et 1,87.]
3. Déterminer les équations des projections sur les plans de coordonnées de l'intersection du cône $x^2+y^2=z/2$ et du cylindre $(y-1)^2+(z+4)^2=1$ représentés en figure 9.
4. Déterminer les équations des projections sur les plans de coordonnées des deux surfaces
$$x^2+y^2+z^2=1 \quad \text{et} \quad x^2y^2+y^2z^2+z^2x^2=2xyz.$$
Factoriser les équations obtenues, en déduire l'intersection.
5. D'un point $P$ on mène les tangentes $PA$, $PB$ à un cercle de rayon $R$. Trouver la courbe parcourue par le centre de gravité du triangle $PAB$ lorsque le point $P$ décrit une droite située à distance $d$ du centre du cercle.

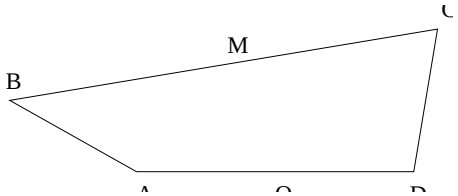



FIGURE 10 Le mécanisme de WATT.

6. On donne la cubique
$$x = \frac{t^2}{1+t^2}, \qquad y = \frac{t^3}{1+t^2}.$$
Une droite quelconque $D$ la rencontre en trois points $A$, $B$, et $C$. Les tangentes à la courbe en ces points rencontrent à nouveau la courbe en trois autres points $\alpha$, $\beta$, $\gamma$. Démontrer que les points $\alpha$, $\beta$, $\gamma$ sont alignés. [Cet exercice ne nécessite pas beaucoup de variables, mais le temps de calcul est très élevé.]
7. On considère la courbe de l'espace définie par les équations paramétriques
$$x = t^4, \qquad y = t^3, \qquad z = t.$$
Déterminer une condition sur les paramètres pour que quatre points de la courbe soient coplanaires. Déterminer une condition pour que trois points de la courbe soient alignés.
8. La courbe de WATT. À la fin du XVIII$^{\text{e}}$ siècle, on recherchait des moyens de transformer un mouvement circulaire (produit par un moteur à vapeur) en un mouvement longitudinal. Le mécanisme représenté en figure 10 fut considéré par J. WATT. Il s'agit d'un quadrilatère articulé $ABCD$. Le côté $AD$ est fixe et on cherche dans ces conditions la courbe parcourue par le point $M$ milieu de $BC$. Pour simplifier on suppose ici $AB = CD = l$, on prend l'origine au milieu de $AD$ sur lequel on prend l'axe des $x$ et on pose $BM = r$. Une fois l'équation de la courbe trouvée, la tracer pour $l = 1$, $AD = 4$, et $r = 3/2$.
9. On considère la courbe d'équation
$$(x^2 + y^2)(ax + by) + xy = 0.$$
   (1) Trouver une paramétrisation ;
   (2) trouver une relation entre les paramètres de trois points alignés ;
   (3) déterminer les points d'inflexion.
10. La normale en $M$ à une parabole $y^2 - 2px = 0$ rencontre l'axe de la parabole en $N$ et la tangente au sommet de la parabole en $P$. Déterminer la courbe parcourue par le milieu de $PN$. [Indication : le vecteur directeur de la normale en $(x, y)$ à une courbe $f(x, y) = 0$ est $(f'_x, f'_y)$.]
11. Déterminer les droites situées sur la surface
$$x = u + v, \qquad y = \frac{1}{u} + \frac{1}{v}, \qquad z = u^2 + v^2.$$

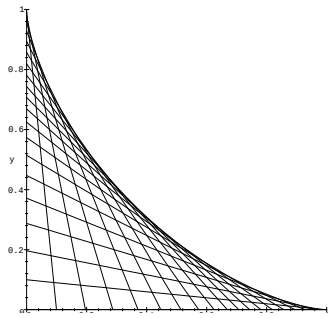


FIGURE 11

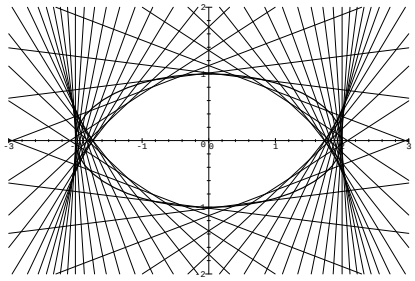


FIGURE 12

12. Calculer l'enveloppe de la famille de cercles dont le centre est situé sur la parabole $y^2 = 2px$ et qui sont tangents à l'axe de la parabole. [On peut simplifier le problème en commençant par paramétrer la parabole.]
13. On considère les plans variables $Q_\lambda$ d'équation $3\lambda^2 x + 3\lambda y + z + \lambda^3 = 0$ et $P$ celui de ces plans qui correspond à $\lambda = a$. Déterminer l'enveloppe de la droite d'intersection de $P$ et de $Q_\lambda$.
14. On considère la glissade d'une échelle contre un mur perpendiculaire au sol (fig. 11). Déterminer l'équation de l'enveloppe des positions de l'échelle. Cette courbe s'appelle une *astroïde*.
15. Déterminer l'enveloppe des droites perpendiculaires aux diamètres d'une ellipse aux extrémités de ceux-ci (fig. 12). Par diamètre on entend toute corde qui passe par le centre.

CHAPTER VII

# Suites réelles

Les suites pour lesquelles le calcul formel a le plus à offrir sont les suites qui satisfont une récurrence linéaire. Nous considérons d'abord celles-ci, en détaillant le cas des coefficients constants où la récurrence peut être résolue, bien que ce soit rarement la chose à faire. Nous poursuivons par les récurrences d'ordre un, en montrant sur quelques exemples détaillés comment une approche purement empirique permet dans certains cas d'obtenir toute l'information désirable sur une suite. Un cas particulier important de solutions de récurrences d'ordre un est fourni par les sommes et les produits que nous abordons ensuite, et nous concluons le chapitre par des indications sur la façon de mener le calcul de valeurs d'une suite, ou d'évaluer numériquement la limite d'une suite, car l'approche numérique est bien souvent la seule dont on dispose.

## 1. Récurrences linéaires

Les récurrences linéaires se prêtent à des calculs qui sont impossibles sur des récurrences plus générales. Nous détaillons une partie de ces opérations dans cette section, lorsque les coefficients sont constants ou polynomiaux [†].

**1.1. Coefficients constants.** Les récurrences linéaires avec des coefficients qui ne dépendent pas de l'indice ont le double avantage d'être faciles à résoudre et à manipuler et d'être très fréquentes en pratique.

1.1.1. *Résolution*

Il est possible de résoudre ces récurrences en utilisant le polynôme caractéristique. La commande `rsolve` effectue cette résolution ; en voici la syntaxe :

```
rsolve(u(n+2)-3*u(n+1)+u(n)=0,u(n));
```

$$\frac{\sqrt{5}}{5(3+\sqrt{5})}(-3u_1+7u_0-\sqrt{5}u_1+3\sqrt{5}u_0)\left(\frac{2}{3+\sqrt{5}}\right)^n - \frac{\sqrt{5}}{5(-3+\sqrt{5})}(3u_1-7u_0-\sqrt{5}u_1+3\sqrt{5}u_0)\left(-\frac{2}{-3+\sqrt{5}}\right)^n$$

[†]Pour se faire une idée d'autres opérations sur ces récurrences et comprendre pourquoi elles sont si importantes pour le calcul formel, le lecteur peut jouer avec le *package* `gfun`.

Lorsque les valeurs initiales ne sont pas précisées, le système les exprime de façon symbolique (ici $u_0, u_1$).

Dès que l'ordre de la récurrence est supérieur ou égal à 5, le polynôme caractéristique peut ne pas être résoluble par radicaux. L'expression de la suite fait alors intervenir des nombres algébriques non explicites. C'est ce qui se produit par exemple avec la récurrence suivante.

$$u_{n+5} = 3u_{n+2} + u_n, \qquad u_0 = u_1 = u_2 = u_3 = u_4 = 1, \tag{1}$$

pour laquelle Maple renvoie un résultat peu utilisable, qui pourrait d'ailleurs être amélioré en remplaçant la limite de $(z-R)/(-1+3z^3+z^5)$ par $1/(9R^2+5R^4)$.

```
rec:=u(n+5)=3*u(n+2)+u(n):
ini:={u(0)=1,u(1)=1,u(2)=1,u(3)=1,u(4)=1}:
rsolve({rec} union ini,u(n));
```

$$\sum_{\_R=\%1} \left( -\frac{\left(\lim_{z\to \_R} \frac{(z-\_R)(2z^3-1-z-z^2+2z^4)}{-1+3z^3+z^5}\right)\left(\frac{1}{\_R}\right)^n}{\_R} \right)$$

$$\%1 := \mathrm{RootOf}(-1+3\_Z^3+\_Z^5)$$

1.1.2. *Calcul rapide*

La résolution explicite de la récurrence ne donne pas un accès facile aux valeurs de la suite. Par exemple, la suite définie par la récurrence (1) ne prend que des valeurs entières, mais il est difficile d'obtenir $u_{100}$ à partir de la solution ci-dessus.

En réalité, pour calculer des valeurs de la suite, il vaut mieux ne pas essayer de résoudre la récurrence. Des méthodes rapides seront décrites en §4.

Dans le cas des récurrences linéaires à coefficients constants, il est en outre possible de calculer très rapidement des valeurs de la suite pour des indices très élevés. En effet, le calcul de $u_n$ nécessite moins de $n$ opérations arithmétiques pour $n$ suffisamment grand ! Pour réaliser ce calcul, la récurrence est d'abord récrite sous forme matricielle. Dans l'exemple de la récurrence (1), la forme matricielle équivalente est

$$\begin{bmatrix} u_{n+5} \\ u_{n+4} \\ u_{n+3} \\ u_{n+2} \\ u_{n+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 3 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} u_{n+4} \\ u_{n+3} \\ u_{n+2} \\ u_{n+1} \\ u_n \end{bmatrix},$$

équation que l'on peut encore noter $U_{n+1} = A \cdot U_n$. En considérant ainsi des vecteurs, la récurrence est devenue une récurrence linéaire d'ordre un et sa solution est donc $U_n = A^n U_0$. Or le calcul de $A^n$ nécessite moins de $n$ multiplications de matrices, comme on l'a vu au chapitre IV. C'est ainsi par

exemple que fonctionne la procédure `fibonacci` de Maple (voir chap. II pour la façon d'afficher le code d'une procédure Maple).

Pour une récurrence donnée, on pourra s'inspirer de la procédure suivante qui calcule des valeurs de la récurrence (1)[†]:

```
p:=proc(n:nonnegint)
local A,res,i,l;
   A := array([[0,0,3,0,1],[1,0,0,0,0],[0,1,0,0,0],[0,0,1,0,0],[0,0,0,1,0]]);
   res := array([1,1,1,1,1]);
   l := convert(n−4,base,2);
   if l[1] = 1 then res := array([4,1,1,1,1]) fi;
   for i in subsop(1 = NULL, l) do
      A := evalm(A*A);
      if i = 1 then res := evalm(A&*res) fi
   od;
   res[1]
end:
```

Pour $n = 100$, cette procédure est trois fois plus rapide que l'évaluation du résultat donné par `rsolve`, et cinquante fois pour $n = 1000$.

### 1.1.3. *Fonction génératrice et comportement asymptotique*

Tout comme le calcul de valeurs de la suite, la détermination du comportement asymptotique ne nécessite pas une résolution explicite de la récurrence. Le comportement asymptotique de la suite se lit sur sa *fonction génératrice*. La série génératrice d'une suite $u_n$ est la série entière

$$u(z) = \sum_{n\geq 0} u_n z^n. \tag{2}$$

Lorsque la série génératrice a un rayon de convergence non nul, ce qui est le cas ici, on parle de *fonction* génératrice.

Pour une récurrence linéaire à coefficient constants, cette fonction est toujours une fraction rationnelle. Elle s'obtient facilement en multipliant tous les termes de la récurrence par $z^n$ et en sommant par rapport à $n$. L'équation obtenue est linéaire en la série génératrice. La commande `ztrans` de Maple calcule $u(1/z)$. Voici par exemple pour les nombres de Fibonacci

```
subs([u(0)=0,u(1)=1],ztrans(u(n+2)=u(n+1)+u(n),n,z)):
f:=normal(subs(z=1/z,solve(",ztrans(u(n),n,z))));
```

$$f := -\frac{z}{-1+z+z^2}$$

La décomposition en éléments simples d'une fraction rationnelle sur $\mathbb{C}$ consiste à écrire la fraction rationnelle comme une somme finie de termes de la forme

$$\frac{a_{\alpha,k}}{(1-\frac{z}{\alpha})^k}.$$

[†]On peut aussi utiliser le *package* `gfun`.

Le $n^{\text{e}}$ coefficient de Taylor de ce terme à l'origine vaut

$$a_{\alpha,k}\alpha^{-n}\binom{n+k-1}{k-1}.$$

Le $n^{\text{e}}$ élément de la suite, qui est aussi le $n^{\text{e}}$ coefficient de Taylor de $u(z)$ d'après (2), est donc égal à la somme des contributions de chacun des termes de la décomposition en éléments simples. Pour obtenir le comportement asymptotique, il suffit donc de se concentrer sur ceux de ces termes pour lesquels $\alpha$ est de plus petit module et de ne calculer que les coefficients $a_{\alpha,k}$ correspondants. C'est en cela que le calcul nécessite moins d'opérations que la résolution exacte qui a besoin de tous les coefficients. Les $\alpha$ sont les racines du dénominateur de la fonction génératrice (inverses des racines du polynôme caractéristique) et les coefficients $a_{\alpha,k}$ sont obtenus en calculant la valeur du numérateur de la fraction rationnelle en $\alpha$ et la première dérivée non-nulle du dénominateur en ce point.

Exemple 1. Pour les nombres de Fibonacci, les racines du dénominateur sont données par

```
fsolve(denom(f),z);
```

$$-1.618033989, \quad 0.6180339887$$

C'est donc la seconde racine qui a le plus petit module. Comme le dénominateur est de degré 2, chacune de ces racines est un pôle simple de la fonction génératrice. D'où le coefficient :

```
alpha:="[2]:
a:=subs(z=alpha,-numer(f)/alpha/diff(denom(f),z)):
```

Avec ces valeurs, le $n^{\text{e}}$ nombre de Fibonacci vaut asymptotiquement $a/\alpha^n$ (ici $\alpha$ est l'inverse du nombre d'or). En outre, ce comportement asymptotique est très rapidement utilisable numériquement, comme on le voit par comparaison avec les valeurs exactes de la fonction `fibonacci` du *package* `combinat` :

```
a/alpha^10,combinat[fibonacci](10);
```

$$55.00363618, \quad 55$$

Exemple 2. Pour la récurrence

$$u_{n+5} = 3u_{n+2} + u_n, \qquad u_0 = u_1 = u_2 = u_3 = u_4 = 1,$$

la fonction génératrice vaut

```
rec:=u(n+5)=3*u(n+2)+u(n):
ini:={u(0)=1,u(1)=1,u(2)=1,u(3)=1,u(4)=1}:
f:=normal(subs(z=1/z,solve(subs(ini,ztrans(rec,n,z)),
    ztrans(u(n),n,z))));
```

$$f := \frac{-1 - z - z^2 + 2z^3 + 2z^4}{-1 + 3z^3 + z^5}$$

et les modules des racines sont

```
sol:=fsolve(denom(f),z,complex):
map(abs,[sol]);
```

$$\left[0.7065263561,\ 0.7065263561,\ 1.738902813,\ 1.738902813,\ 0.662510317\right]$$

Là encore, les pôles sont simples et c'est le cinquième qui a le plus petit module.

```
alpha:=sol[5]:
a:=subs(z=alpha,-numer(f)/alpha/diff(denom(f),z)):
```

Le $n^{\text{e}}$ élément de la suite vaut donc asymptotiquement $a/\alpha^n$, estimation qui se vérifie numériquement (`p` est la procédure de la p. 167) :

```
evalf(p(100)),a/alpha^100;
```

$$0.2645979826\,10^{18}, \qquad 0.2648334906\,10^{18}$$

L'approximation est moins bonne que pour les nombres de FIBONACCI parce que le module suivant des racines du dénominateur n'est pas très loin : l'erreur relative est en $(0,66/0,70)^n$, soit deux millièmes pour $n = 100$, ce que confirme la comparaison ci-dessus.

## 1.2. Coefficients polynomiaux.

### 1.2.1. *Résolution*

Contrairement au cas des coefficients constants, on ne sait pas résoudre toutes les récurrences linéaires à coefficients polynomiaux. Il est néanmoins facile de déterminer si une équation comme

$$nu_{n+2} - 5u_{n+1} - (n+1)u_n = 0 \tag{3}$$

a des solutions qui sont des polynômes. La commande `rsolve` de Maple ne le fait malheureusement pas. Ce calcul s'effectue en étudiant les coefficients du membre gauche de l'équation lorsque $u_n$ est un polynôme en $n$ de degré $\alpha$. Le degré maximal est $\alpha + 1$. Le coefficient de $n^{\alpha+1}$ est 0. Ensuite, le coefficient de $n^\alpha$ vaut $2\alpha - 6$ et doit être nul. Le degré vaut donc $\alpha = 3$. Maple peut conclure la résolution par une méthode de coefficients indéterminés :

```
rec:=n*u(n+2)-5*u(n+1)-(n+1)*u(n):
pol:=n->a3*n^3+a2*n^2+a1*n+a0:
collect(eval(subs(u=pol,rec)),n);
```

$$(-3\,a_3 - 2\,a_2)\,n^2 + (-4\,a_1 - 7\,a_3 - 6\,a_2)\,n - 6\,a_0 - 5\,a_1 - 5\,a_3 - 5\,a_2$$

```
factor(subs(solve({coeffs(",n)},{a0,a1,a2,a3}),pol(n)));
```

$$a_1\,n(2n-1)(n-1)$$

Une fois trouvée une solution, une seconde solution indépendante s'obtient par variation de la constante. Nous verrons comment mener à bien ce calcul en exercice 7.

On peut également calculer les solutions rationnelles. Maple n'a pas de procédure pour cela, mais nous allons montrer comment procéder sur la récurrence suivante :

$$\begin{aligned} &n(n+3)(n^2+3n+3)u_{n+2} \\ &\qquad - 5(n+2)(n^2+n+1)u_{n+1} - (n+1)^2(n^2-n+1)u_n = 0. \end{aligned} \tag{4}$$

La première étape consiste à localiser les pôles de la fonction $u$, ou de manière équivalente, il s'agit de déterminer le dénominateur $D$ de $u$. Si $\alpha$ est une racine de $D$, alors $\alpha-1$ est un pôle de $u(n+1)$ et $\alpha-p$ est un pôle de $u(n+p)$. Le terme droit de l'équation (4) étant le polynôme nul, le terme gauche ne peut avoir de pôle. En conséquence, sauf si des racines de $D$ diffèrent d'un entier, le coefficient $a_k(n)$ de $u_{n+k}$ doit être un multiple de $D(n+k)$. Dans ce cas, $D$ doit diviser $\tilde{D} = \operatorname{pgcd}(a_0(n), a_1(n-1), \ldots, a_p(n-p))$. Il suffit donc de poser $u_n = N_n/\tilde{D}(n)$ et de rechercher les solutions polynomiales de la nouvelle équation, comme ci-dessus. Reste le cas où des racines de $D$ diffèrent d'un entier, mais alors cet entier ne peut pas être plus grand que la plus grande différence entière entre les racines de $a_0$ et celles de $a_p$. L'exercice 6 montre comment se traite ce cas.

Sur notre exemple, la résolution est très simple :

```
rec:=n*(n+3)*(n^2+3*n+3)*u(n+2)-5*(n+2)*(n^2+n+1)*u(n+1)
     -(n+1)^2*(n^2-n+1)*u(n):
d:=gcd(coeff(rec,u(n)),gcd(subs(n=n-1,coeff(rec,u(n+1))),
       subs(n=n-2,coeff(rec,u(n+2)))));
```

$$d := n^3 + 1$$

Ce polynôme est un multiple du dénominateur. Le changement de variable $u(n) = a(n)/d(n)$ donne alors :

```
map(normal,eval(subs(u=(n->(a(n)/(n^3+1))),rec)));
```

$$na(n+2) - 5a(n+1) - (n+1)a(n)$$

et l'on retrouve l'équation (3) dont on connaît les solutions polynomiales. Les solutions rationnelles de (4) sont donc de la forme

$$C\frac{n(2n-1)(n-1)}{n^3+1}.$$

Une autre solution indépendante peut se déduire de la seconde solution de (3) déterminée à l'exercice 7.

### 1.2.2. *Fonctions génératrices*

Les récurrences linéaires et les équations différentielles linéaires sont liées. Les coefficients d'une série entière solution d'une équation différentielle linéaire à coefficients polynomiaux satisfont une récurrence linéaire. Inversement la série génératrice d'une solution d'une récurrence linéaire à coefficients polynomiaux satisfait une équation différentielle linéaire. Le passage de l'une à l'autre enrichit l'étude de chacune d'elles.

EXEMPLE 3. La suite $a_n$ est définie par $a_{n+2} = a_{n+1} + \frac{2}{n+2}a_n$, $a_0, a_1$ réels. La commande `rsolve` n'est pas capable de résoudre cette récurrence, mais la fonction génératrice de cette suite est simple. Le calcul de cette fonction génératrice est très facile à réaliser avec le *package* `gfun`. Nous montrons ci-dessous comment faire avec la commande `ztrans`. Au lieu de chercher une

équation différentielle en la fonction génératrice

$$y(z) = \sum_{n\geq 0} a_n z^n,$$

`ztrans` incite à en chercher une satisfaite par $u(z) = y(1/z)$ :

```
rec:=a(n+2)=a(n+1)+2*a(n)/(n+2):
subs(ztrans(a(n),n,z)=u(z),
     ztrans(numer(op(1,rec)-op(2,rec)),n,z));
```

$$-z^3\frac{\partial}{\partial z}u(z) - a(1)z - zu(z) + z^2\frac{\partial}{\partial z}u(z) + a(0)z - 2u(z)$$

Maple peut alors résoudre cette équation différentielle :

```
dsolve(",u(z));
```

$$u(z) = -\frac{z}{4}\left(-5a(1)z^2 + 5a(0)z^2 + 6a(1)z - 6a(0)z - 2a(1)\right.$$
$$\left. +2a(0) - 4e^{-2/z}z^2\_C1\right)/(-1+3z-3z^2+z^3)$$

```
y:=map(factor,collect(subs(z=1/z,op(2,")),_C1));
```

$$y := -\frac{\_C1e^{-2z}}{(-1+z)^3} + \frac{1}{4}\frac{(5-6z+2z^2)(-a(1)+a(0))}{(-1+z)^3}$$

Il reste à utiliser les conditions initiales pour déterminer $\_C_1$ :

```
subs(_C1=solve(coeff(series(y,z),z,0)=a(0),_C1),y);
```

$$-\frac{e^{-2z}(-\frac{5}{4}a(1)+\frac{9}{4}a(0))}{(-1+z)^3} + \frac{1}{4}\frac{(5-6z+2z^2)(a(1)+a(0))}{(-1+z)^3}$$

La solution de la récurrence se déduit de cette expression : le développement en série de $\exp(-2z)$ est connu et le fait de diviser une série par $1-z$ revient à en sommer les coefficients. En conclusion,

$$a_n = \frac{a_1-a_0}{8}(n+2)(n+5) + \frac{9a_0-5a_1}{8}\sum_{k=0}^{n}\frac{(-2)^k(n-k+1)(n-k+2)}{k!}.$$

En guise de vérification, nous calculons ci-dessous quelques valeurs numériques. Il est très peu efficace d'utiliser la fonction `sum` pour calculer la somme, en particulier parce qu'on ne s'intéresse qu'à sa valeur numérique. On utilisera donc plutôt la fonction `seq` suivie d'un `convert`.

```
rec1:=proc(n) option remember; rec1(n-1)+2*rec1(n-2)/n end:
rec1(0):=a0: rec1(1):=a1:
rec2:=proc(n) local k;
   (a1-a0)/8*(n+2)*(n+5)+(9*a0-5*a1)/8*
   convert([seq((-2)^k/k!*(n-k+1)*(n-k+2),k=0..n)],`+`)
end:
seq(rec1(i),i=0..5);
seq(rec2(i),i=0..5);
```

$$a0, a1, a1+a0, \frac{5}{3}a1+a0, \frac{13}{6}a1+\frac{3}{2}a0, \frac{17}{6}a1+\frac{19}{10}a0$$
$$a0, a1, a1+a0, \frac{5}{3}a1+a0, \frac{13}{6}a1+\frac{3}{2}a0, \frac{17}{6}a1+\frac{19}{10}a0$$

Dans le cas des récurrences linéaires à coefficients non polynomiaux, on ne sait pas faire grand chose, si ce n'est pour les récurrences d'ordre un qui sont traitées ci-après.

### 1.3. Exercices.

1. Résoudre $u_{n+2} = ku_{n+1}^3/u_n$. [Indication : prendre le logarithme].
2. On considère le système
$$\begin{cases} u_{n+1} = u_n + 2v_n \\ v_{n+1} = u_n + v_n \end{cases}$$
avec $u_0 = v_0 = 1$. Calculer $\sum u_n \frac{t^n}{n!}$, $\sum v_n \frac{t^n}{n!}$.
3. Déterminer le comportement asymptotique de la suite définie par
$$u_0 = u_1 = u_2 = 1 \quad u_{n+3} = 7u_{n+2} + u_n.$$
4. Déterminer les deux premiers termes du développement asymptotique de la suite définie par
$$u_0 = u_3 = 1 \quad u_1 = u_2 = 0 \quad u_{n+4} - 3u_{n+3} + u_{n+2} + 4u_n = 0.$$
5. Déterminer le comportement asymptotique de la suite définie par
$$u_0 = u_1 = u_2 = u_3 = u_4 = 1 \quad u_{n+5} = 3u_{n+2} + u_n.$$
[Il faut prendre en compte les contributions de deux pôles].
6. On considère la récurrence
$$\begin{aligned} \left(n^5 + 7n^4 + 18n^3 + 21n^2 + 9n\right) u_{n+2} \\ + \left(-6n - 3n^4 - 9n^3 - 9n^2\right) u_{n+1} \\ + \left(1 - n^5 + n^3 - n^2\right) u_n = 0. \end{aligned}$$
   (1) Vérifier que certaines des racines du coefficient de tête et certaines des racines du coefficient de queue diffèrent d'un entier ;
   (2) en déduire un entier $p$ tel que deux racines du dénominateur d'une solution rationnelle ne puissent avoir une différence entière plus grande que $p$ ;
   (3) construire une récurrence satisfaite par $v_n = u_{pn}$ ;
   (4) vérifier que les racines du dénominateur d'une solution rationnelle de cette nouvelle récurrence ne peuvent différer d'un entier ;
   (5) résoudre cette récurrence ;
   (6) en déduire les solutions rationnelles de l'équation de départ.
7. Le but de cet exercice est de déterminer une seconde solution indépendante de la récurrence (3) :
$$nu_{n+2} - 5u_{n+1} - (n+1)u_n = 0$$

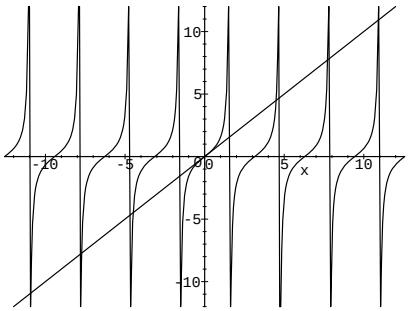



FIGURE 1
*Les points fixes de la tangente.*

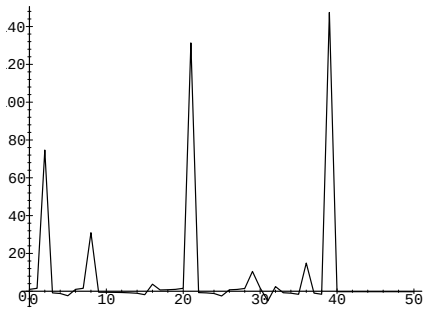



FIGURE 2
*Les premières valeurs de la suite.*

connaissant la solution $p_n = n(2n-1)(n-1)$. La variation de la constante consiste à chercher une solution de la forme

$$p_n \sum_{k=0}^{n-1} t_k.$$

Pour cela

(1) Déterminer la récurrence linéaire d'ordre 1 satisfaite par la suite $t_n$ ;
(2) chercher les solutions rationnelles de cette récurrence.

La somme demeure symbolique.

## 2. Récurrences d'ordre un

Plus les récurrences sont générales, moins les outils permettant de les étudier sont puissants. Les théorèmes les plus généraux sur les suites monotones bornées ou sur les suites de CAUCHY peuvent parfois s'appliquer et la vérification des hypothèses nécessaires à l'application de ces théorèmes peut parfois être facilitée par le calcul formel.

**2.1. Récurrences du type $u_{n+1} = f(u_n)$.** Les problèmes liés à ces suites sont d'une grande richesse et sont loin d'être tous résolus. En pratique, et pour des suites réelles, il faut étudier le comportement de la fonction $f$ au voisinage de ses *points fixes*, solutions de l'équation $x = f(x)$. Si elle y est *contractante*, ce qui se traduit par une dérivée de valeur absolue strictement inférieure à 1, alors toute suite passant dans un certain domaine autour du point fixe va converger ; le point fixe est dit *attractif*. Si au contraire la dérivée a une valeur absolue strictement supérieure à 1, le point fixe est *répulsif* : seules les suites passant exactement par le point fixe y convergent. Il existe aussi des points fixes *indifférents*, où la dérivée a une valeur absolue exactement 1 et où une étude plus précise de la suite est nécessaire avant de pouvoir conclure.

EXEMPLE 4. Voici un exemple où tous les points fixes sont répulsifs : la suite définie par $u_{n+1} = \tan(u_n)$. Cette suite a pour points fixes les solutions

de $x = \tan(x)$, abscisses des points d'intersection sur la figure 1. Cette figure s'obtient par :

```
plot({tan(x),x},x=-4*Pi..4*Pi,-12..12,discont=true);
```

Si $u_0$ est l'un de ces points, la suite est constante et donc converge. En revanche, le comportement des autres suites est gouverné par la stabilité de ces points fixes, donc par la dérivée de la fonction tangente en leur voisinage. Comme $\tan' = 1 + \tan^2$, la dérivée en $u_0$ est strictement supérieure à 1 lorsque $u_0 \neq k\pi$. Le seul cas à considérer est $u_0 = k\pi$ et là, nullité de la dérivée seconde et signe de la dérivée troisième permettent de conclure. Par conséquent la suite ne converge jamais en dehors des points fixes dont les premières valeurs sont :

```
seq(fsolve(tan(x)=x,x,(2*k+1)*Pi/2..(2*k+3)*Pi/2),k=-1..9);
```

$$0, 4.493409458, 7.725251837, 10.90412166, 14.06619391, 17.22075527,$$
$$20.37130296, 23.5194525, 26.66605426, 29.81159879, 32.95638904$$

Pour les autres valeurs de $u_0$, le comportement de la suite ne présente aucune régularité. Les premières valeurs lorsque $u_0 = 1$ sont représentées en figure 2 :

```
u[0]:=1.:for i to 50 do u[i]:=tan(u[i-1]) od:
plot([seq([i,u[i]],i=0..50)]);
```

Exemple 5. On veut résoudre :

$$\sqrt{x + 2\sqrt{x + 2\sqrt{\cdots + 2\sqrt{3x}}}} = x.$$

De manière équivalente, il s'agit de déterminer pour quelles valeurs initiales $u_0 = x$ la suite définie par $u_{n+1} = \sqrt{x + 2u_n}$ converge vers $x$. Les points fixes sont donnés par

```
solve(l=sqrt(x+2*l),l);
```

$$\mathrm{RootOf}\left(-x - 2\_Z + \_Z^2\right)$$

Et la réponse est parmi

```
solve(x=",x);
```

$$0, \quad 3$$

Il reste à vérifier que les deux suites ayant ces valeurs initiales convergent bien vers la limite souhaitée, ce qui est aisé puisqu'elles s'avèrent constantes.

Exemple 6. La suite $u_{n+1} = \sin(u_n)$. Les points fixes sont maintenant les solutions de $x = \sin(x)$. La commande `solve` de Maple trouve la racine 0 :

```
solve(sin(x)=x,x);
```

$$0$$

Après une itération de la suite, $u_1$ se trouve dans l'intervalle $[-1, 1]$ où il est facile de prouver que 0 est le seul point fixe. L'intervalle $[-1, 1]$ est lui-même strictement inclus dans $]-\pi/2, \pi/2[$ à l'intérieur duquel la dérivée du sinus est toujours inférieure à 1 (sauf en 0). La suite converge donc vers 0 pour

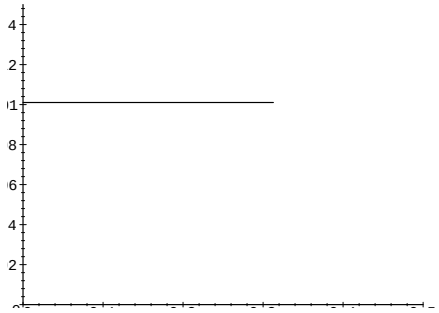



FIGURE 3
`plot(essai,0..0.5,0..0.015)`

toute valeur initiale. Il est également possible d'étudier à quelle vitesse la suite converge vers 0. Nous le ferons au chapitre VIII.

**2.2. Récurrences du type $u_{n+1} = f(n, u_n)$.** En dehors des récurrences d'ordre un appartenant à des classes plus restreintes et que nous avons déjà vues, on ne peut pas donner de règle générale applicable face à une telle récurrence. L'exemple que nous traitons en détail dans cette section illustre l'approche empirique que l'on doit souvent adopter.

Il s'agit d'étudier la récurrence

$$u_{n+1} = \frac{e^{u_n}}{n+1}$$

en fonction de $u_0$ (réel positif).

Intuitivement, lorsque $u_n$ est trop grand, l'exponentielle le fait "exploser" et la suite $u_n$ diverge rapidement vers $+\infty$. Inversement, si $e^{u_n}$ reste limité alors que $n$ croît, $u_n$ devrait tendre vers 0. Le programme suivant permet d'effectuer quelques expérimentations numériques :

```
essai:=proc(u0)
local u,n;
  u:=u0;
  for n to 100 do u:=evalf(exp(u)/n) od
end:
```

Pour $u_0$ plus petit qu'environ $0,3$ le programme renvoie toujours une même valeur, à savoir approximativement $0,010\,102\,579\,55$. Pour des valeurs de $u_0$ supérieures à $0,3$, le système renvoie un message d'erreur invoquant un argument trop grand pour le calcul de l'exponentielle. Il semble donc exister une valeur seuil de $u_0$. La figure 3 p. 175 confirme cette impression.

L'existence de cette valeur seuil est facile à prouver : l'exponentielle étant croissante, en notant $u_n^{(x)}$ et $u_n^{(x')}$ les suites correspondant à deux valeurs initiales $x$ et $x'$, on a $x < x' \iff u_n^{(x)} < u_n^{(x')}$ pour tout $n$. Donc si la suite diverge pour une certaine valeur initiale $u_0$, elle diverge pour toutes les

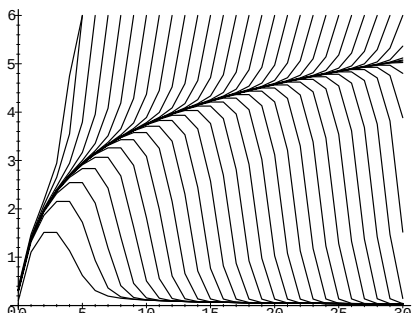



FIGURE 4

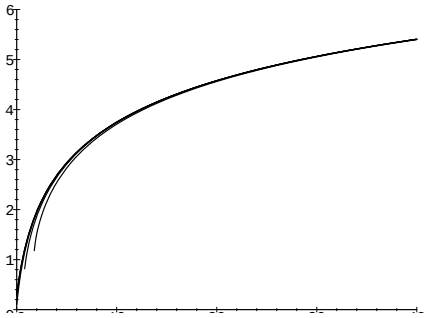



FIGURE 5

valeurs supérieures. Inversement si elle converge pour une valeur initiale $u_0$, elle reste bornée pour toutes les valeurs inférieures. Pour achever la preuve de l'existence de ce seuil, il reste à exhiber une valeur pour laquelle la suite diverge et une autre pour laquelle elle converge, ce qui peut se faire en minorant une suite divergente et en majorant une suite convergente, mais cela ne met pas en œuvre le calcul formel.

En revanche, la valeur de ce seuil peut être calculée, ainsi que bien d'autres informations relatives à cette suite. La figure 4 montre les premiers termes de la suite pour différentes valeurs initiales. En abscisse sont portés les indices et en ordonnée les valeurs. Les points d'une même suite sont reliés entre eux. Plusieurs questions se posent naturellement au vu de cette figure :

(1) Toutes les suites semblent "sortir" d'une même courbe. Quelle est cette courbe ?
(2) Presque toutes les valeurs initiales utilisées pour tracer ce dessin sont dans un intervalle de largeur $0,02$ autour de la valeur seuil. Comment calculer précisément cette valeur seuil ?
(3) Toutes les suites qui ne divergent pas semblent converger. Peut-on le prouver ?
(4) Toutes les suites qui ne divergent pas semblent "aboutir" à une même courbe. Peut-on préciser ce qu'est cette courbe (au moins asymptotiquement) ?
(5) Comment ce dessin a-t-il été obtenu ?

La clé de la réponse aux questions (1), (2), (3) et (5) est l'observation suivante : si $u_n \leq u_{n-1}$, alors $u_{n+1} < u_n$. Autrement dit, soit la suite est croissante, soit elle commence par croître, puis ne peut que décroître. Comme par ailleurs elle reste positive, elle est alors convergente. Les courbes de la partie inférieure de la figure sont obtenues en recherchant les paliers, qui sont solutions de $u_{n+1} = u_n$. Pour $n < 2$, les deux courbes $y = \exp(x)/(n+1)$ et $y = x$ n'ont pas d'intersection, il ne peut donc se produire de palier. Ensuite, les deux courbes ont deux intersections (l'une à gauche et l'autre à

droite de 1), mais il est impossible que $u_n < 1$ pour $n \geq 2$, ce qui détermine la solution qui nous intéresse.

La fonction $W$ de Maple, définie par $W(x)\exp[W(x)] = x$, permet de donner une forme "explicite" à l'ordonnée du $n^{\text{e}}$ palier :

```
solve(exp(z)/(n+1)=z,z);
```

$$-W\left(-\frac{1}{n+1}\right)$$

Il faut faire attention à ce que la bonne solution soit choisie. Comme la racine carrée ou le logarithme, la fonction $W$ a plusieurs branches. Une vérification numérique convainc que la solution renvoyée par `solve` n'est pas la bonne pour ce problème :

```
evalf(subs(n=2,"));
```

$$0.6190612866$$

On vérifiera que la bonne racine est en fait $-W(-1,-1/(n+1))$. La courbe obtenue en traçant cette fonction pour $n$ réel entre 0 et 40 passe par les points de départ des paliers horizontaux de la figure 4. Ensuite, on "remonte" aux valeurs antérieures de la suite, ce qui donne le dessin de la figure 5 :

```
t[0]:=-W(-1,-1/(x+1)):
for i to 5 do t[i]:=log((x+1)*subs(x=x+1,t[i-1])) od:
plot({seq(t[i],i=0..5)},x=0..40,0..6);
```

Ceci répond à la question (1) : la courbe de départ est celle que l'on obtient comme limite des courbes $t_i$ et la convergence est très rapide. Le procédé employé permet également de répondre à la question du seuil : il suffit de prendre une valeur de $u_n$ pas trop loin de la courbe et de "remonter" à $u_0$. La convergence est extrêmement rapide :

```
for x in [2,4,8,16,32] do
   y:=-W(-1,-1/(x+1));
   for j from x by -1 to 1 do y:=log(j*y) od:
   w[x]:=evalf(y)
od:
seq(w[x],x=[2,4,8,16,32]);
```

$$0.1013550039, 0.3044035424, 0.3132473044, 0.3132776390, 0.3132776397$$

Les détails de la réponse à la dernière question seront abordés dans l'exercice 2, il reste donc la question (4). La méthode que nous allons employer est en fait très générale et s'adapte à de nombreuses situations de calcul de développements asymptotiques. On cherche un développement asymptotique $z(n)$ d'une suite tendant vers 0 telle que $\exp(z(n-1))/n$ ait le même développement asymptotique. Il suffit de raffiner l'estimation $z = 0$ :

```
f:=exp(z)/n:
0:
to 5 do asympt(eval(subs(z=subs(n=n-1,"),f)),n) od;
```

$$\frac{1}{n}$$
$$\frac{1}{n}+\frac{1}{n^2}+\frac{3}{2n^3}+\frac{13}{6n^4}+\frac{73}{24n^5}+O\left(\frac{1}{n^6}\right)$$
$$\frac{1}{n}+\frac{1}{n^2}+\frac{5}{2n^3}+\frac{20}{3n^4}+\frac{437}{24n^5}+O\left(\frac{1}{n^6}\right)$$
$$\frac{1}{n}+\frac{1}{n^2}+\frac{5}{2n^3}+\frac{23}{3n^4}+\frac{641}{24n^5}+O\left(\frac{1}{n^6}\right)$$
$$\frac{1}{n}+\frac{1}{n^2}+\frac{5}{2n^3}+\frac{23}{3n^4}+\frac{665}{24n^5}+O\left(\frac{1}{n^6}\right)$$

Avec $k$ itérations, cette méthode fournit un développement correct jusqu'au terme en $1/n^k$ inclus. À titre de confirmation, voici la valeur numérique de ce dernier développement en $n = 100$ :

```
subs(n=100.,");
```

$$0.01010257944 + O(0.000000000001)$$

À $10^{-10}$ près, il s'agit de la valeur renvoyée par la procédure `essai` du début de cette section.

### 2.3. Exercices.

1. On étudie la suite définie par $u_0 > 0$,
$$u_{n+1} = \frac{u_n - \log(1+u_n)}{u_n^2}.$$
   (1) Vérifier que la fonction $x \mapsto (x - \log(1+x))/x^2$ ne possède qu'un point fixe strictement positif ;
   (2) calculer une approximation numérique du point fixe ;
   (3) vérifier que le point fixe est attractif ;
   (4) déterminer le développement asymptotique de $u_n$, en représentant le point fixe symboliquement.
2. Cet exercice détaille les étapes nécessaires à l'obtention de la figure 4 p. 176.
   (1) Écrire une procédure qui prend en argument une valeur de $u_0$ et qui renvoie la liste $[0, u_0, 1, u_1, \ldots, 30, u_{30}]$, sauf si $u_k > 6$ pour un certain $k$ auquel cas elle renvoie $[0, u_0, 1, u_1, \ldots, k-1, u_{k-1}, k, 6]$ ;
   (2) calculer les valeurs des ordonnées des 20 premiers paliers ;
   (3) calculer les valeurs de $u_0$ correspondantes ;
   (4) tracer l'ensemble des listes produites par la procédure du (1) sur ces valeurs de $u_0$ : on doit obtenir la partie inférieure de la figure ;
   (5) pour obtenir la partie supérieure de la figure, ajouter $2/k$ à l'ordonnée obtenue en (2) pour le palier d'abscisse $k$, calculer la valeur de $u_0$ correspondante et la liste de valeurs par la procédure du (1) ;
   (6) il n'y a plus qu'à tracer toutes ces listes ensemble.
3. On s'intéresse à la suite définie par $u_n = n^{1/u_{n-1}}$ et $u_0 = 1$, dont on veut obtenir le comportement asymptotique.

(1) Calculer les premières valeurs ;
(2) faire un dessin des valeurs obtenues ;
(3) ce dessin suggère d'étudier les deux sous-suites $v_n = u_{2n}$ et $w_n = u_{2n+1}$. Les dessiner ;
(4) écrire la récurrence reliant $u_n$ à $u_{n-2}$ sous la forme $u_n = f(u_{n-2})$ ;
(5) remplacer $1/u_{n-2}$ par 0 dans cette expression, puis en calculer le développement asymptotique ;
(6) remplacer $u_{n-2}$ dans $f$ par le développement obtenu à la question précédente, en ayant soin d'y remplacer $n$ par $n-2$ ;
(7) répéter cette dernière étape et observer que l'expression converge vers le développement asymptotique de $u_n$.

## 3. Sommes et produits

Les sommes et les produits sont des cas particuliers de suites définies par une récurrence linéaire d'ordre 1 : si $S_n$ et $P_n$ sont définis par

$$S_n = \sum_{k=0}^{n} u_k, \quad \text{et} \quad P_n = \prod_{k=0}^{n} u_k,$$

alors ils vérifient

$$S_n - S_{n-1} = u_n \quad \text{et} \quad P_n = u_n P_{n-1}.$$

Cependant, les particularités des sommes et des produits permettent de les étudier par des méthodes spécifiques.

Dans cette section, nous décrivons surtout les calculs de sommes exactes, c'est-à-dire les cas où il est possible de donner une formule pour $S_n$ où $k$ n'intervient plus. Nous reviendrons sur d'autres aspects des sommes dans le chapitre suivant sur les séries.

Quant au produit, on peut le ramener à une somme en prenant le logarithme, tout en faisant attention aux signes.

**3.1. Sommes géométriques.** Les sommes géométriques sont les sommes de la forme $\sum_{i=a}^{b} x^i$. Maple sait les reconnaître et les évaluer :

```
sum(x^i,i=a..b);
```

$$\frac{x^{b+1}}{x-1} - \frac{x^a}{x-1}$$

Ces sommes sont très importantes en pratique. En particulier, elles interviennent naturellement dans les calculs d'intérêts composés. Nous détaillons en exercice quelques questions que l'on peut se poser sur le plan épargne logement.

**3.2. Suites hypergéométriques et sommes indéfinies.** Les suites hypergéométriques sont une généralisation des suites géométriques : dans une suite géométrique, le quotient $u_{n+1}/u_n$ est une constante indépendante de $n$, alors que dans une suite hypergéométrique, ce quotient est une fraction rationnelle en $n$. Des exemples de suites hypergéométriques sont la factorielle ou les coefficients binomiaux.

Cette notion est importante pour le calcul de *sommes indéfinies*, qui sont aux sommes ce que les primitives sont aux intégrales. Autrement dit la suite $\{S_n\}$ est une somme indéfinie de la suite $\{u_n\}$ lorsque $S_n - S_{n-1} = u_n$ pour tout $n$. Bien sûr, la connaissance d'une somme indéfinie permet le calcul de sommes définies (avec des bornes) ; il suffit de soustraire les valeurs de la somme indéfinie aux deux bornes.

Un algorithme important dû à Gosper détermine si une suite hypergéométrique admet une somme qui est elle-même une suite hypergéométrique. Cet algorithme forme la base de la fonction `sum` des systèmes de calcul formel. Ainsi, lorsque la fonction `sum` de Maple ne trouve pas de somme à une suite hypergéométrique, cela signifie que l'on peut *prouver* qu'il n'existe pas de somme qui soit hypergéométrique.

Voici un exemple de succès de l'algorithme :

```
v:=(3*n^3-10*n^2-9*n+11)*(n+1)!*2^n/(2*n+3)!/(n-5)/(n-4):
Sum(v,n)=sum(v,n);
```

$$\sum_n \frac{(3n^3-10n^2-9n+11)(n+1)!\,2^n}{(n-4)(n-5)(2n+3)!} = -\frac{2^{n-2}n!\,(3n-1)}{(n-5)(2n+1)!}$$

Voici un exemple où l'échec de `sum` prouve qu'il n'y a pas de somme hypergéométrique :

```
sum(1/n!,n);
```

$$\sum_n \frac{1}{n!}$$

**3.3. Autres sommes indéfinies.** Il y a très peu de sommes que l'on sait exprimer sous forme explicite. En dehors de celles que nous venons de voir, la jungle des sommes est très foisonnante et on ne peut pas donner de règle générale. Nous donnons en exercice deux exemples dans lesquels la chance nous sourit.

**3.4. Exercices.**

1. Le plan d'épargne logement.
   (1) Pour un taux annuel de 6,32 %, vérifier que le taux mensuel correspondant vaut environ 0,51199983 % ;
   (2) pour une somme empruntée de 100 000 F sur 8 ans, vérifier que les versements mensuels sont environ de 1 321,17 F ;
   (3) calculer les intérêts versés pendant les neuf années civiles durant lesquelles le prêt s'étend, sachant que le prêt démarre en mars ;
   (4) si 20 % de ces intérêts sont déductibles des impôts dans une limite de 15 000 F chaque année pendant les 6 premières années, quel est le taux réel du prêt, c'est-à-dire le taux du prêt correspondant à l'argent réellement remboursé ?
   (5) quelle est le montant du prêt qui minimise ce taux ?

   Note : dans la réalité, il faut aussi tenir compte des frais de notaire et de l'assurance du prêt...

2. Soit $x \in \mathbb{R}$, que vaut la limite de la suite $u_n$ suivante ?

$$u_n = (1 + x + x^2 + \cdots + x^9)(1 + x^{10} + \cdots + x^{90}) \cdots (1 + x^{10^n} + \cdots + x^{9\cdot 10^n}).$$

3. La suite $u_n$ étant définie par $u_{n+2} = 3u_{n+1} - u_n$ et $u_1 = 2$, $u_2 = 5$, que vaut la somme
$$\sum_{n=1}^{N} \operatorname{arccot}(u_n) \ ?$$

4. Avec $x \in [0, 1]$, on demande d'étudier
$$u_p = \frac{1}{p} \sum_{k=0}^{p} \cos^{2k}(k\pi x).$$

## 4. Calculs numériques

Dans les cas que nous avons traités jusqu'ici, les récurrences étaient linéaires ou d'ordre un et les valeurs de la suite se calculaient donc sans difficulté. Nous montrons dans cette section comment calculer ces valeurs pour des récurrences plus complexes.

**4.1. Premiers termes d'une suite récurrente.** Les programmes suivants calculent les premières valeurs de la suite d'origine combinatoire définie par

$$u_{n+1} = u_n + \sum_{i=0}^{n} u_i u_{n-1-i}, \qquad u_0 = 1.$$

Les commentaires que nous allons faire sur ces programmes et sur leurs performances doivent permettre de guider l'écriture de programmes pour d'autres récurrences.

4.1.1. *Programme 1 : méthode naïve*

```
u1:=proc(n:nonnegint) local i;
   if n=0 then 1
   else u1(n−1)+convert([seq(u1(i)*u1(n−1−i),i=0..n−1)],§'+')
   fi
end:
```

Ce programme[†] fonctionne récursivement en s'arrêtant lorsque $n$ vaut 0. Il a cependant l'inconvénient de recalculer très souvent les mêmes termes : le nombre $a_n$ d'appels à la procédure `u1` pour calculer $u_n$ vérifie

$$a_n = 1 + a_{n-1} + 2\sum_{i=0}^{n-1} a_i, \qquad a_0 = 1,$$

[†]La fonction `sum` ici serait inappropriée puisque son but premier est de trouver une somme indéfinie. La méthode vraiment naïve consisterait à utiliser la fonction `sum`.

formule à partir de laquelle on peut montrer que ce nombre croît en $(2+\sqrt{3})^n$. Le temps de calcul croît donc de manière *exponentielle* par rapport à $n$. La première ligne du tableau 1 (p. 184) montre le temps de calcul de $u_n$ par cette procédure pour diverses valeurs de $n$.

4.1.2. *Programme 2 : option remember*

```
u2:=proc(n:nonnegint) local i;
option remember;
   u2(n−1)+convert([seq(u2(i)*u2(n−1−i),i=0..n−1)],§‘+‘)
end:
u2(0):=1:
```

Ce programme est bien plus efficace que le précédent. Le gain principal vient de l'utilisation de l'`option remember`. En effet, chaque $u_n$ n'est maintenant calculé qu'une fois (voir chap. II). Comme chaque $u_n$ nécessite $n$ multiplications et $n$ additions, le temps de calcul est devenu *polynomial* par rapport à $n$. La contrepartie à ce gain en vitesse est une perte en mémoire : tous les nombres $u_k$ calculés sont gardés en mémoire. Cependant dans de nombreuses applications, on a besoin de tous les éléments d'une suite et dans ce cas, cette façon de programmer la récurrence est recommandée.

4.1.3. *Programmes 3 et 4: plus spécialisé*

À partir de la fonction génératrice de la suite, on peut montrer que la suite $u_n$ satisfait aussi une récurrence linéaire :

$$u_n = \frac{(6n-3)u_{n-1} - (n-2)u_{n-2}}{n+1}, \qquad u_0 = 1, u_1 = 2.$$

Donc chaque terme peut se calculer à partir des deux précédents au lieu de les utiliser tous. Ceci conduit à un programme simple :

```
u3:=proc(n:nonnegint)
local i,u0,u1,u2;
   u1 := 1; u2 := 2;
   for i from 2 to n do
      u0 := u1; u1 := u2; u2 := (2*u0−3*u1+(6*u1−u0)*i)/(i+1)
   od;
   u2
end:
u3(0):=1: u3(1):=2:
```

L'intérêt principal de ce programme est que chaque $u_n$ est calculé en faisant un nombre fixe d'opérations à partir des précédents. Le temps de calcul est donc devenu *linéaire* par rapport à $n$ (tant que les nombres manipulés sont assez petits pour que l'on puisse considérer le coût des opérations comme constant). Par ailleurs, si l'on veut calculer un seul $u_n$ et que l'on n'a pas besoin des $u_k$ pour $0 \leq k < n$, alors ce programme effectue le calcul sans gaspiller de mémoire. Il est toutefois plus fréquent que l'on veuille calculer

tous les éléments d'une suite jusqu'à un certain rang et on aura alors recours au programme suivant.

```
u4:=proc(n:nonnegint)
option remember;
   (2*u4(n−2)−3*u4(n−1)+(6*u4(n−1)−u4(n−2))*n)/(n+1)
end:
u4(0):=1:u4(1):=2:
```

4.1.4. *Programmes 1bis à 4bis : calculs approchés*

Dans les programmes précédents, lorsque $n$ est grand une grosse partie du temps de calcul provient des opérations sur des grands entiers ($u_{1000}$ a 761 chiffres décimaux). Si l'on accepte de se contenter de valeurs approchées, les opérations élémentaires vont se faire en temps constant (indépendant des nombres manipulés) ce qui peut s'avérer très payant sur des grands nombres.

Pour obtenir des valeurs approchées des éléments de la suite, il suffit de modifier les lignes définissant $u_0$ et $u_1$ dans ces programmes par l'ajout d'un point (`.`) après le 1 et le 2. Les nombres entiers 1 et 2 deviennent alors les nombres "flottants" (nombres décimaux en virgule flottante) `1.0` et `2.0` et cette propriété se propage aux sommes et produits auxquels ils participent. Pour d'autres récurrences, il vaut mieux mettre explicitement `evalf` dans le corps de la procédure, car la propagation des nombres flottants aux fonctions n'est pas totale.

4.1.5. *Programme 3ter : flottants machine*

La commande `evalhf` permet de tirer parti des nombres flottants du processeur de la machine. Ses performances dépendent fortement de l'architecture de la machine, mais peuvent être excellentes. En particulier, Maple utilise `evalhf` pour tracer ses dessins. Il est parfois possible d'appliquer `evalhf` sur une procédure. Lorsque c'est le cas, tous les calculs intermédiaires sont faits rapidement. Le programme 1 ci-dessus ne peut pas être exécuté par `evalhf` :

```
evalhf(u1(10));
```

Error, unable to evaluate function convert in evalhf

L'application de `evalhf` sur les programmes utilisant l'option *remember* est fatale : le système meurt. Il reste donc le programme 3, pour lequel `evalhf` fonctionne et donne de bons résultats (tab. 1). Cependant, les flottants de la machine sont souvent limités en taille (précisions dans la documentation de `evalhf[constants]`). Ainsi, dans notre exemple et sur notre machine, nous ne pouvons pas évaluer $u_n$ par `evalhf(u3(n))` pour $n \geq 405$.

4.1.6. *Comparaison*

Le tableau 1 montre le temps de calcul de $u_n$ par les différents programmes pour les valeurs de $n$ indiquées en haut de chaque colonne. Pour rendre ces résultats indépendants de la machine sur laquelle les tests ont été effectués,

TABLE 1 Les temps de calcul des programmes.

| $n$ | 2 | 5 | 10 | 20 | 50 | 100 | 500 | 1 000 | 5 000 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0,154 | 7,056 | 5 411 | — | — | — | — | — | — |
| 1bis | 0,097 | 7,637 | 5 797 | — | — | — | — | — | — |
| 2 | 0,013 | 0,057 | 0,150 | 0,530 | 2,583 | 11,53 | 1 572 | 12 882 | 72 645 |
| 2bis | 0,020 | 0,061 | 0,165 | 0,688 | 3,705 | 16,47 | 484,8 | 2 112 | 67 861 |
| 3 | 0,004 | 0,023 | 0,034 | 0,088 | 0,200 | 0,493 | 6,798 | 33,27 | 450,4 |
| 3bis | 0,013 | 0,037 | 0,085 | 0,195 | 0,390 | 0,795 | 4,239 | 9,078 | 51,34 |
| 3ter | 0,005 | 0,007 | 0,011 | 0,017 | 0,022 | 0,047 | — | — | — |
| 4 | 0,014 | 0,032 | 0,086 | 0,196 | 0,448 | 0,990 | 10,00 | 40,11 | 1 367 |
| 4bis | 0,011 | 0,044 | 0,125 | 0,290 | 0,798 | 1,307 | 7,273 | 15,70 | 107,0 |

l'unité de temps dans ce tableau est le dixième du temps nécessaire au calcul de $u_{500}$ par le programme 4. Les cases manquantes n'ont pu être obtenues, faute de temps ou d'espace mémoire. Ce tableau permet de tirer plusieurs enseignements sur la programmation en Maple.

La linéarité du temps de calcul n'est vraiment atteinte que pour les programmes 3bis et 4bis qui opèrent sur des flottants. Elle est également bien approchée par les programmes 3 et 4 tant que les entiers manipulés ne sont pas trop gros, c'est-à-dire de moins de 100 chiffres.

Les flottants en Maple sont lents par rapport aux entiers tant que ces derniers ne sont pas trop gros : avant la colonne $u_{500}$ les programmes "bis" sont plus lents que les programmes exacts. Ceci provient de ce que les flottants sont représentés de manière interne comme des couples d'entiers (mantisse, exposant) et les calculs sont en réalité effectués sur des entiers. En revanche, lorsque `evalhf` peut être appliqué, il utilise les flottants de la machine et ceci mène à de bien meilleures performances (ligne 3ter).

Mais l'enseignement principal de ce tableau, c'est qu'il est bien plus important d'améliorer l'algorithme de calcul que sa programmation. Ainsi le programme 3 et le programme 4 ont des performances comparables, alors que les passages du programme 1 au programme 2 puis au programme 3 ont des conséquences spectaculaires sur les temps de calcul. Le fait d'avoir remarqué que la suite satisfaisait une récurrence linéaire en a rendu possible le calcul pour de très grandes valeurs de l'indice. Si la récurrence linéaire avait été à coefficients constants, il aurait été possible d'être encore bien plus rapide, en appliquant la méthode du §1.1.

**4.2. Évaluations numériques de limites.** Pour estimer numériquement la limite d'une suite qui converge, il faut bien sûr savoir calculer des éléments de la suite ; les indications données sur l'exemple ci-dessus permettent de traiter la plupart des cas rencontrés dans la pratique. Pour évaluer la limite, le calcul des valeurs de la suite pour des indices très élevés est avantageusement remplacé par une technique simple d'accélération de convergence due à ROMBERG. Si $u_n$ est la suite, et si elle converge vers sa limite en admettant

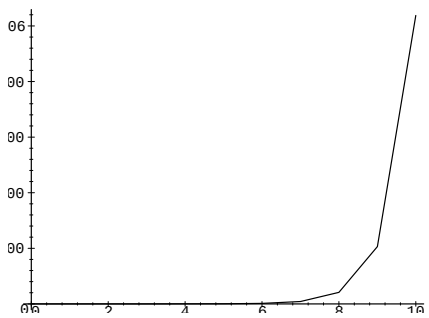



FIGURE 6
`plot([seq([i,u4bis(i)],i=0..10)])`

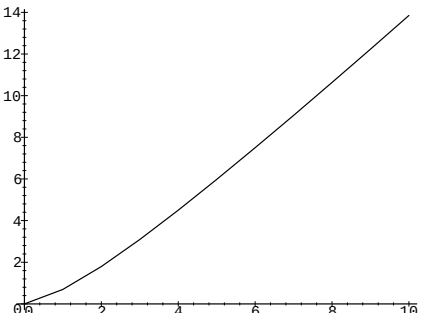



FIGURE 7
`plot([seq([i,log(u4bis(i))],i=0..10)])`

un développement asymptotique du type

$$u_n = \ell + \frac{a}{n} + \frac{b}{n^2} + \cdots, \tag{5}$$

alors la méthode de ROMBERG consiste à calculer des valeurs de $u_n^{(1)} = 2u_{2n} - u_n$. Sans connaître la valeur de $a$, il est ainsi éliminé et asymptotiquement

$$u_n^{(1)} = \ell + \frac{c}{n^2} + \cdots$$

converge plus vite vers $\ell$ que $u_n$. Une nouvelle étape de la même technique conduit à calculer des valeurs de $u_n^{(2)} = \frac{4}{3}u_{2n}^{(1)} - \frac{1}{3}u_n^{(1)}$, et ainsi de suite.

La figure 6 p. 185 représente les premières valeurs de la suite $u_n$ de la section précédente. Leur croissance rapide conduit à s'intéresser à la suite $\log u_n$ qui est représentée en figure 7 p. 185. Cette dernière figure suggère qu'asymptotiquement $\log u_n \sim cn$, où la pente $c$ est à déterminer. Les premières valeurs de la suite obtenues par

```
seq(log(u3bis(2^i))/2^i,i=0..7);
```

forment la première colonne du tableau 2 p. 186 et confirment la convergence (assez lente) de cette suite. Dans ce tableau, les premières valeurs des différentes suites sont indiquées. Ces valeurs ont été obtenues de la manière suivante, `u[j,n]` représentant $u_{2^n}^{(j)}$ :

```
for i from 0 to 7 do u[0,i]:=ln(u3bis(2^i))/2^i od:
for j to 7 do for i from 0 to 7-j do
     u[j,i]:=(2^j*u[j-1,i+1]-u[j-1,i])/(2^j-1) od od:
```

Comme on le voit sur le tableau 2 (où les valeurs ont été arrondies à 4 décimales), la convergence ne paraît pas avoir été beaucoup accélérée par le procédé : il faut regarder la diagonale du tableau, c'est-à-dire

```
seq(u[j,7-j],j=0..7);
```

1.704170344, 1.746640265, 1.751918544, 1.753464028, 1.754082842,
1.754362332, 1.754495423, 1.754560397

TABLE 2 Premier essai d'accélération.

| $n$ | $u_n$ | $u_n^{(1)}$ | $u_n^{(2)}$ | $u_n^{(3)}$ | $u_n^{(4)}$ | $u_n^{(5)}$ | $u_n^{(6)}$ | $u_n^{(7)}$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0,6931 | 1,0986 | 1,4392 | 1,6160 | 1,6935 | 1,7292 | 1,7462 | 1,7546 |
| 2 | 0,8959 | 1,3540 | 1,5939 | 1,6887 | 1,7281 | 1,7460 | 1,7545 | |
| 4 | 1,1250 | 1,5339 | 1,6768 | 1,7256 | 1,7454 | 1,7544 | | |
| 8 | 1,3294 | 1,6411 | 1,7195 | 1,7442 | 1,7541 | | | |
| 16 | 1,4853 | 1,6999 | 1,7411 | 1,7535 | | | | |
| 32 | 1,5926 | 1,7308 | 1,7519 | | | | | |
| 64 | 1,6617 | 1,7466 | | | | | | |
| 128 | 1,7042 | | | | | | | |

Dans ce cas particulier, cette diagonale de valeurs ne montre pas une convergence convaincante.

La raison pour laquelle le procédé n'a pas fonctionné de façon spectaculaire est que la suite $\log u_n/n$ ne converge pas comme dans l'équation (5). La suite $u_n$ satisfaisant une récurrence linéaire, on peut cependant s'attendre (voir chap. VIII) à ce qu'elle se comporte asymptotiquement comme

$$u_n \sim r^n n^\alpha \left(a_0 + \frac{a_1}{n} + \frac{a_2}{n^2} + \cdots\right). \tag{6}$$

Dans ces conditions, après une étape du procédé, le développement obtenu est :

```
asympt(expand(log(r^n*n^alpha*(a[0]+a[1]/n+a[2]/n^2)))/n,n,3):
asympt(2*subs(n=2*n,")-",n,3):
map(expand,");
```

$$\ln(r) + \frac{\alpha \ln(2)}{n} - \frac{a_1}{2a_0 n^2} + O\left(\frac{1}{n^3}\right)$$

Autrement dit, après une étape du procédé de ROMBERG sur la suite $\{\log u_n\}$, la situation est ramenée à celle de l'équation (5). Le tableau 3 est alors calculé comme suit :

```
for i from 0 to 7 do v[0,i]:=ln(u3bis(2^i))/2^i od:
for i from 0 to 6 do v[1,i]:=2*v[0,i+1]-v[0,i] od:
for j from 2 to 7 do for i from 0 to 7-j do
    v[j,i]:=(2^(j-1)*v[j-1,i+1]-v[j-1,i])/(2^(j-1)-1) od od:
```

La diagonale montre alors une vraie convergence. On peut par ailleurs montrer que la limite est $\log(3+2\sqrt{2}) \simeq 1,762\ 747\ 174$. L'erreur de $0,06$ pour $u_{128}$ est donc transformée en une erreur de près de $4.10^{-8}$ ($u_1^{(7)} \simeq 1,762\ 747\ 134$), à partir des *mêmes 128 premiers éléments* de la suite.

### 4.3. Exercices.

1. Le programme `u3` effectue trois affectations en trop : on peut éviter la dernière itération de la boucle. Récrire le programme en supprimant ces trois affectations et comparer alors au temps de calcul du programme `u4` pour les petites valeurs de $n$.

TABLE 3 Second essai d'accélération.

| $n$ | $u_n$ | $u_n^{(1)}$ | $u_n^{(2)}$ | $u_n^{(3)}$ | $u_n^{(4)}$ | $u_n^{(5)}$ | $u_n^{(6)}$ | $u_n^{(7)}$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0,6931 | 1,0986 | 1,6094 | 1,7486 | 1,7613 | 1,7627 | 1,7627 | 1,7627 |
| 2 | 0,8959 | 1,3540 | 1,7138 | 1,7597 | 1,7626 | 1,7627 | 1,7627 | |
| 4 | 1,1249 | 1,5339 | 1,7483 | 1,7622 | 1,7627 | 1,7627 | | |
| 8 | 1,3294 | 1,6411 | 1,7587 | 1,7627 | 1,7627 | | | |
| 16 | 1,4853 | 1,6999 | 1,7617 | 1,7627 | | | | |
| 32 | 1,5926 | 1,7308 | 1,7625 | | | | | |
| 64 | 1,6617 | 1,7466 | | | | | | |
| 128 | 1,7042 | | | | | | | |

2. Vérifier que les coefficients à utiliser pour la $k^{\text{e}}$ itération de ROMBERG sont
$$u_n^{(k)} = \frac{2^k}{2^k-1} u_{2n}^{(k-1)} - \frac{1}{2^k-1} u_n^{(k-1)}.$$
3. On considère la suite $u_n = n!/((n/e)^n\sqrt{n})$. Il n'est pas très difficile de voir que la suite converge et la fonction `limit` de Maple détermine que sa limite est $\sqrt{2\pi} \simeq 2,506\ 628\ 274$. Appliquer les méthodes de cette section pour retrouver cette valeur numérique.
4. L'exemple de la suite $u_n$ de cette section peut être poursuivi : une fois que l'on connaît $r$ dans l'équation (6), on peut diviser $u_n$ par $r^n$ et chercher à déterminer numériquement $\alpha$. Sachant que $\alpha$ est un rationnel très simple, on peut alors diviser aussi par $n^\alpha$, et déterminer numériquement $a_0$. On doit trouver ainsi une approximation numérique de
$$a_0 = \frac{\sqrt{4+3\sqrt{2}}}{2\sqrt{\pi}}.$$
Cet exemple sera repris p. 216.

CHAPTER VIII

# Séries et développements asymptotiques

Peu de problèmes admettent une solution exacte. Cependant, des développements en série ou des développements asymptotiques fournissent souvent des informations approchées assez précises qui sont inaccessibles par d'autres voies.

Nous commençons ce chapitre par des problèmes pratiques de calcul numérique à partir d'une série, puis nous montrons comment obtenir des développements en série ou des développements asymptotiques de suites, d'intégrales, ou de solutions d'équations. Dans ce domaine, il est essentiel de *justifier* les opérations purement formelles que le logiciel effectue. Les systèmes de calcul formel ne mettent en effet aucune entrave à l'interversion d'un signe somme et d'un signe intégrale et une utilisation aveugle peut mener à des résultats faux. Nous insistons dans ce chapitre sur les calculs, et renvoyons le lecteur aux ouvrages d'analyse pour les problèmes de justification.

## 1. Séries numériques

**1.1. Calcul approché de constantes définies par des séries.** Deux types de problèmes bien distincts se posent lors du calcul de valeurs numériques à partir d'une série. Le premier est de déterminer combien de termes de la série doivent être utilisés et avec quelle précision ils doivent être calculés. Le second problème est celui de l'efficacité : les calculs qui ne peuvent être évités doivent être effectués au moindre coût.

Les problèmes d'efficacité surgissent surtout lors du calcul de plusieurs milliers de décimales, où des méthodes spécialisées sont employées. Ce domaine très intéressant sort cependant du cadre de ce livre ; les calculs que nous allons faire sont tous limités à quelques centaines de décimales.

Exemple 1. Le sinus intégral de Fresnel est défini par

$$S(x) = \int_0^x \sin\Big(\frac{\pi}{2}t^2\Big)\, dt. \tag{1}$$

L'intégration terme à terme du développement du sinus de l'intégrande donne le développement en série suivant :

$$S(x) = \sum_{n=0}^{\infty} \frac{(-1)^n(\pi/2)^{2n+1}}{(2n+1)!\,(4n+3)} x^{4n+3}, \tag{2}$$

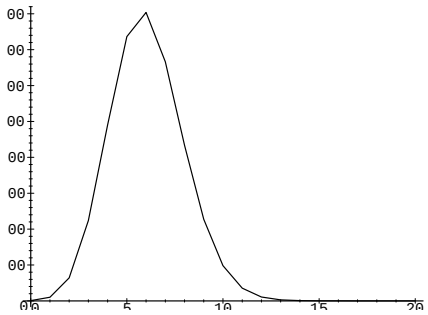



FIGURE 1
`plot([seq([n,(Pi/2)^(2*n+1)/(2*n+1)!/(4*n+3)*3^(4*n+3)],n=0..20)])`

formule qui permet le calcul des valeurs de $S(x)$ avec une précision arbitraire pour tout réel $x$. Bien entendu, la commande `evalf(FresnelS(x))` calcule ces valeurs numériques. Cependant nous allons montrer ici comment écrire une procédure prenant en argument $x$ et renvoyant une estimation de $S(x)$ en tenant compte de la valeur de `Digits` (le nombre de décimales correctes souhaitées), car la méthode à employer est typique de ce que l'on doit faire pour calculer une valeur approchée à partir de telles séries.

Le cœur de la procédure consiste en une boucle qui calcule les termes de la série et les additionne, en essayant de faire le moins d'opérations possible. Dans le cas de $S(x)$, cette boucle s'écrit ainsi :

```
t:=evalf(x^2*Pi/2);
u:=t^2;
t:=x*t;
s:=t/3;
olds:=0;
for k from 2 by 2 while s<>olds do
    olds:=s;
    t:=-u*t/(k^2+k);
    s:=s+t/(2*k+3)
od;
```

Pour finir l'écriture de la procédure, il reste à régler le problème de la précision à utiliser dans les calculs intermédiaires, si le nombre de chiffres corrects que doit avoir le résultat est `Digits`. Les valeurs absolues des premiers termes de la somme pour $x = 3$ sont tracés en figure 1. Comme souvent pour les séries convergentes, les termes commencent par croître en valeur absolue avant de décroître pour tendre vers 0. Du point de vue de l'évaluation, cela pose un problème : pour obtenir $k$ décimales du résultat, il faut calculer les premiers termes avec plus de précision. Ainsi, dans notre exemple le $4^{\text{e}}$ terme de la série vaut environ 13 326 alors que $S(3) \simeq 0,496\ 313$ ; pour obtenir cette valeur avec 10 décimales, il faut donc calculer les premiers termes de la suite

avec 15 chiffres significatifs. La précision à utiliser pendant le calcul dépend donc de la taille du terme maximal et de la taille du résultat, informations généralement inaccessibles *avant* le calcul.

Dans l'exemple qui nous occupe, nous verrons en §2.3 comment montrer que le logarithme du terme maximal est de l'ordre de $\pi x^2/2$. C'est un exercice plus délicat que de montrer que cette estimation est toujours par excès. D'autre part, à partir de l'intégrale il n'est pas difficile de voir que $S$ est de l'ordre de $1/2$ lorsque $x$ tend vers l'infini. Un nombre de décimales égal à $\texttt{Digits} + \log_{10}(\pi x^2/2)$ convient donc.

Un autre problème à résoudre est celui des erreurs d'arrondi. Si chaque terme de la série est calculé avec une erreur $\epsilon$, l'erreur accumulée au bout de $N$ termes est $N\epsilon$ dans le cas le pire. Pour obtenir une erreur inférieure à $\eta$ sur le résultat, il faudrait donc calculer tous les termes de la série avec une erreur inférieure à $\eta/N$, mais il faut donc connaître la valeur de $N$ avant la fin du calcul.

Comme les termes de la somme sont de signe alternant et décroissent à partir d'un certain rang, le reste est majoré par le premier terme négligé (s'il est dans la partie décroissante). Il suffit donc d'évaluer un ordre de grandeur de la valeur de $N$ pour laquelle le $N^{\text{e}}$ terme est de l'ordre de $\epsilon$. Nous verrons en §3.2 comment déterminer que $N \simeq \frac{1}{2}\log(1/\epsilon)\log\log(1/\epsilon)$ convient, et ce indépendamment de $x$ (pour $\epsilon$ très proche de 0). De ceci découle qu'il faut rajouter au nombre de décimales utilisées pendant le calcul environ $\log_{10}(\texttt{Digits}) + c$, où $c$ est une constante peu élevée (3 convient).

Pour conclure, il suffit donc de rajouter avant les instructions données plus haut les lignes

```
S:=proc(x)
local s,u,t,olds,k,oldDigits;
   oldDigits:=Digits;
   Digits:=Digits+ceil(.196+.868*evalhf(ln(x)))+length(Digits)+3;
```

et de terminer la procédure par

```
   evalf(s,oldDigits)
end:
```

**1.2. Évaluation exacte de séries.** Bien qu'il soit très rare que la somme d'une série s'exprime à l'aide des fonctions usuelles, une telle somme, lorsqu'elle est calculable, permet d'accéder à des propriétés globales de la série difficiles d'accès autrement. Outre le cas des séries pour lesquelles on sait trouver une somme indéfinie dont il n'y a plus qu'à calculer la limite, nous donnons ici quelques exemples de séries qui se somment. Bien souvent, la série géométrique est à la base du calcul, ainsi parfois que quelques autres séries entières élémentaires.

Les suites hypergéométriques ont été définies au chap. VII (p. 179), et les séries hypergéométriques sont les sommes de ces suites. Plus précisément, on

note

$$ {}_pF_q\left( \begin{array}{c} a_1,\dots,a_p \\ b_1,\dots,b_q \end{array} \middle| z \right) = \sum_{n\geq 0} \frac{(a_1)_n \cdots (a_p)_n}{(b_1)_n \cdots (b_q)_n} \frac{z^n}{n!}, $$

où $(a)_n = a(a+1)\cdots(a+n-1)$. Lorsque $p \leq q+1$, le rayon de convergence de cette série est non nul et on parle alors de fonction hypergéométrique pour sa somme. Si ces fonctions sont admises comme fonctions de base, alors toutes les fractions rationnelles se somment ; nous en donnerons quelques exemples.

EXEMPLE 2. En regardant $\sin(nx)$ comme $\Im(e^{inx})$, la somme

$$ \sum_{n\geq 1} \frac{\sin(nx)}{n} $$

apparaît comme la primitive d'une série géométrique. Ceci conduit à une forme exacte :

```
evalc(Im(sum(exp(I*x)^n/n,n=1..infinity)));
```

$$ -\arctan\left(-\sin(x), 1-\cos(x)\right) $$

Il faut ensuite un peu de patience avec les fonctions de simplification de Maple avant d'arriver à la forme plus simple

$$ \frac{\pi - x}{2}, $$

valable pour $0 < x < 2\pi$.

EXEMPLE 3. Voici un exemple qui porte sur une fraction rationnelle. Il montre qu'il ne faut pas se limiter à des manipulations purement formelles de séries infinies, mais bien justifier chacune des étapes du calcul. La fonction $f$ ci-dessous admet une décomposition en éléments simples particulièrement élémentaire :

```
f:=1/(4*n+1)/(4*n+2)/(4*n+3);
```

$$ f := \frac{1}{(4n+1)(4n+2)(4n+3)} $$

```
convert(f,parfrac,n);
```

$$ \frac{1}{2(4n+1)} - \frac{1}{2(2n+1)} + \frac{1}{2(4n+3)} $$

Il est donc tentant de conclure que dans la somme $\sum f(n)$, les inverses des entiers impairs apparaissent tous multipliés par 0 et donc que cette somme est nulle. Or c'est une somme de termes positifs. Réarranger les termes de cette façon n'est pas valide, car cela fait intervenir simultanément des termes d'indice $n$ et d'indice $2n$ de la série. En faisant la somme jusqu'à un indice $N$ fixé avant de le faire tendre vers l'infini, ce qui donne le bon résultat, on prend en compte $N$ termes en moins dans la série négative. Le résultat est alors celui que Maple parvient à trouver dans cet exemple :

```
sum(f,n=0..infinity);
```

$$-\frac{\Psi(1/4)}{8}-\frac{\gamma}{4}-\frac{\ln(2)}{2}-\frac{\Psi(3/4)}{8}$$

Dans cette expression $\Psi$ désigne la dérivée logarithmique de la fonction $\Gamma$ d'Euler et $\gamma$ désigne la constante d'Euler. En réalité, cette expression se simplifie pour donner finalement

$$\sum_{n\geq 0}\frac{1}{(4n+1)(4n+2)(4n+3)}=\frac{1}{4}\ln 2.$$

Ce résultat s'obtient également à partir de la décomposition en éléments simples, par une méthode abélienne, qui consiste à multiplier le sommant par $z^{4n}$, pour faire tendre $z$ vers 1 à la fin du calcul. Chacun des termes de la décomposition en éléments simples se ramène une fois encore à une série très proche de celle de $\log(1-z)$. Maple trouve ainsi la plus simple :

```
sum(z^(4*n)/(2*n+1),n=0..infinity);
```

$$\frac{\ln\left(\frac{1+\sqrt{z^4}}{1-\sqrt{z^4}}\right)}{2z^2}$$

et les deux autres en termes de fonctions hypergéométriques :

```
sum(z^(4*n)/(4*n+1),n=0..infinity);
```

$${}_2F_1\left(\begin{matrix}1,1/4\\5/4\end{matrix}\middle|\, z^4\right)$$

```
sum(z^(4*n)/(4*n+3),n=0..infinity);
```

$$\frac{1}{3}{}_2F_1\left(\begin{matrix}1,3/4\\7/4\end{matrix}\middle|\, z^4\right)$$

Malheureusement, Maple n'arrive pas à simplifier ces expressions ni à calculer leur limite pour $z$ tendant vers 1. Ces deux sommes sont pourtant faciles à calculer : des combinaisons linéaires à partir de la série de $\log(1-z)$ et de $\arctan(z)$ fournissent, en jouant sur la parité :

$$\sum_{n=0}^{\infty}\frac{z^{4n}}{4n+1}=\frac{1}{4z}\ln\left(\frac{1+z}{1-z}\right)+\frac{1}{2z}\arctan(z),$$

$$\sum_{n=0}^{\infty}\frac{z^{4n}}{4n+3}=\frac{1}{4z^3}\ln\left(\frac{1+z}{1-z}\right)-\frac{1}{2z^3}\arctan(z).$$

Il ne reste plus qu'à sommer les différentes contributions (et à justifier le procédé) :

```
limit(-ln((1+z^2)/(1-z^2))/4/z^2+ln((1+z)/(1-z))/8/z*(1+1/z^2)
+arctan(z)/4/z*(1-1/z^2),z=1,left);
```

$$\frac{1}{4}\ln 2.$$

Exemple 4. Voici un exemple où la série du logarithme ne suffit plus pour trouver une forme simple à la somme. Il s'agit de la somme suivante, qui rappelle la formule de Stirling :

$$\sum_{n\geq 0}\frac{n^{n+1}}{n!}e^{-ny}y^n.$$

Cette somme suggère de regarder la série entière $\sum_{n\geq 0} n^{n+1}x^n/n!$ où $x = ye^{-y}$. Or, bien que Maple ne reconnaisse pas cette somme, il en connaît une bien proche :

$$W(x) = \sum_{n\geq 0} (-1)^n n^{n-1} \frac{x^n}{n!}, \qquad |x| < 1/e.$$

Les bons signes sont donnés par $-W(-x)$. Le numérateur du sommant doit être multiplié par $n^2$, ce qui revient à dériver la série et à la multiplier par $x$, deux fois de suite, d'où le résultat :

```
normal(simplify(subs(x=y*exp(-y),x*diff(x*diff(-W(-x),x),x))));
```

$$\frac{y}{(1-y)^3}$$

Une autre classe très importante de sommes calculables "exactement" est la classe des séries hypergéométriques. On retrouve ainsi de nombreuses sommes connues comme limite de la série en 1 :

```
sum(1/n!,n=1..infinity),sum(1/n^2,n=1..infinity),
   sum((-1)^n/(2*n+1),n=0..infinity);
```

$$e-1, \quad \frac{\pi^2}{6}, \quad \frac{\pi}{4}$$

La plupart des sommes définies que trouvent les systèmes de calcul formel sont des sommes hypergéométriques.

**1.3. Convergence et divergence des séries.** Peu d'outils sont disponibles pour étudier la convergence et la divergence des séries. Les plus utiles sont la majoration de la somme des valeurs absolues par une série convergente et la décroissance des valeurs absolues pour une série alternée. Nous illustrons sur quelques exemples l'utilisation du calcul formel dans ces problèmes.

Exemple 5. On étudie la convergence de la série $\sum a_n$ où

$$a_n = \int_0^1 1 - (1-t^n)^{1/n}\, dt.$$

Les variations des intégrandes pour les premières valeurs de $n$ (fig. 2) donnent une idée du comportement des $a_n$. Il est facile de prouver que les $a_n$ sont positifs et que l'intégrande est monotone. On souhaite donc majorer la série $a_n$ par une série convergente (le calcul numérique de quelques valeurs de $a_n$ convainc que la série converge). Ces courbes suggèrent que $1-(1-t^n)^{1/n}$ reste très proche de 0 sur la plus grande partie de $[0,1]$, avant de croître très rapidement pour atteindre 1 pour $t = 1$. La majoration se fait donc en deux étapes : d'une part en majorant la valeur de l'intégrande sur la partie proche de 0, d'autre part en majorant la largeur d'un intervalle dans lequel l'intégrande est majoré par 1. Pour cela, on cherche par exemple pour quelle valeur de $t$ l'intégrande vaut $1/n^2$ (donc commence à "décoller") :

```
asympt(solve(1-(1-t^n)^(1/n)=1/n^2,t),n,3);
```

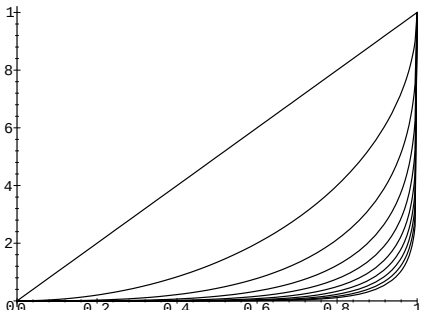



FIGURE 2
`plot({seq(1-(1-t^i)^(1/i),i=1..10)},t=0..1)`

$$1 - \frac{\log n}{n} + O\left(\frac{1}{n^2}\right)$$

L'intégrale est donc majorée par $1/n^2$ sur l'intervalle $[0, 1 - \log n/n]$ et par 1 sur l'intervalle $[1 - 1/n^2, 1]$. Il reste à la majorer sur $[1 - \log n/n, 1 - 1/n^2]$ par la valeur de l'intégrande en $1 - 1/n^2$, c'est-à-dire

```
asympt(subs(t=1-1/n^2,1-(1-t^n)^(1/n)),n,3);
```

$$\frac{\log n}{n} + O\left(\frac{1}{n^2}\right)$$

Ceci permet de majorer $a_n$ par $(2 + \log^2 n)/n^2$ et achève la preuve de la convergence de la série.

EXEMPLE 6. Nature de la série

$$\sum_{n=0}^{\infty} \sin(\pi\sqrt{4n^2+1}).$$

Cette somme est semblable à $\sum \sin(2\pi n)$ qui vaut 0. La périodicité du sinus permet d'obtenir le comportement asymptotique des termes sommés car $\sin(\pi\sqrt{4n^2+1}) = \sin(\pi(\sqrt{4n^2+1} - 2n))$ :

```
asympt(sin(Pi*(sqrt(4*n^2+1)-2*n)),n,3);
```

$$\frac{\pi}{4n} + O\left(\frac{1}{n^3}\right)$$

Donc il existe $N$ tel que pour $n > N$, $\sin(\pi\sqrt{4n^2+1})$ est positif et borné inférieurement par $\pi/(5n)$. Ceci permet de conclure que la série diverge.

EXEMPLE 7. Nature de

$$\sum_{n>1} \frac{(-1)^n \sqrt{n} \sin(1/\sqrt{n})}{n + (-1)^n}.$$

Pour cet exemple, on a envie d'utiliser la convergence des séries alternées dont les termes décroissent en valeur absolue, mais la monotonie pose problème. Voici quelques valeurs :

```
f:=(-1)^n*sqrt(n)*sin(1/sqrt(n))/(n+(-1)^n):
evalf([seq(subs(n=i,f),i=2..17)]);
```

$$\Big[\begin{array}{llll} 0.3062417898, & -0.4726815281, & 0.1917702154, & -0.2417496044, \\ 0.1389218269, & -0.1627266612, & 0.1088107209, & -0.1226980113, \\ 0.08940149714, & -0.09849172063, & 0.0758591436, & -0.08226905886, \\ 0.06587584553, & -0.07063756192, & 0.0582126963, & -0.06188905459 \end{array}\Big]$$

Les signes semblent alterner et il est facile de le prouver. Mais les valeurs absolues des termes ne décroissent pas. En revanche, en groupant $f(n)$ avec $f(n+1)$, on peut prouver la convergence de la somme :

```
f+subs(n=n+1,f);
```

$$\frac{(-1)^n\sqrt{n}\sin(1/\sqrt{n})}{n+(-1)^n}+\frac{(-1)^{n+1}\sqrt{n+1}\sin(1/\sqrt{n+1})}{n+1+(-1)^{n+1}}$$

Maple ne peut pas calculer directement de développement asymptotique de cette expression : il faut préciser la parité de $n$.

```
asympt(subs((-1)^n=1,(-1)^(n+1)=-1,"),n,3);
```

$$-\frac{1}{n^2}+O\left(\frac{1}{n^3}\right)$$

Ceci prouve que la série de terme général $f(2k)+f(2k+1)$ converge ; $f(n)$ tendant vers zéro, la série $\sum f(n)$ converge aussi.

### 1.4. Exercices.

1. La constante de Catalan peut être définie de nombreuses façons. Elle vaut par exemple

$$C=\int_0^1\frac{\arctan x}{x}dx=-\int_0^1\frac{\ln(x)dx}{1+x^2}$$
$$=\int_0^{\pi/2}\frac{x\,dx}{2\sin x}=\int_0^{\pi/2}-\ln(2\sin\frac{x}{2})\,dx.$$

   On peut aussi la représenter comme une somme,

$$C=\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)^2}=\frac{\pi}{8}\log(2+\sqrt{3})+\frac{3}{8}\sum_{n=0}^{\infty}\frac{1}{(2n+1)^2\binom{2n}{n}},$$

   la première de ces deux sommes ayant l'avantage de la simplicité, et la seconde celui de permettre un calcul rapide.

   Écrire une procédure qui calcule cette constante, en prenant le nombre de décimales souhaité en argument.

2. Montrer à l'aide de Maple que

$$\sum_{n=1}^{\infty}\frac{1}{n}\cos^n\theta\cos n\theta=-\ln(\sin\theta),\qquad\theta\in\,]0,\pi[\,,$$
$$\sum_{n=1}^{\infty}\frac{1}{1^2+2^2+\cdots+n^2}=18-24\ln 2.$$

3. La somme double
$$\sum_{n=1}^{\infty}\sum_{m=1}^{\infty}\frac{1}{m^2n+n^2m+2nm}$$
a une valeur rationnelle simple.
   (1) Décomposer le sommant en éléments simples par rapport à $m$ ;
   (2) remarquer qu'une partie indépendante de $m$ se factorise et la somme restante (sur $m$) est un nombre harmonique ($H_n = 1+1/2+\cdots+1/n$) ;
   (3) décomposer le facteur trouvé en éléments simples par rapport à $n$ et annuler deux à deux les contributions provenant du nombre harmonique sauf deux des composants du nombre harmonique ;
   (4) sommer alors les termes restants.

   On pourra vérifier numériquement la valeur trouvée.
4. Calculer
$$\sum_{n=0}^{\infty}\arctan(\frac{1}{1+n+n^2}).$$
[Indication : regarder les premières troncatures.]
5. Calculer les produits
$$\prod_{n\geq 2}\frac{n^3-1}{n^3+1},\quad \prod_{n\geq 0}\left[1+(1/2)^{2^n}\right],\quad \prod_{n\geq 2}\left[1+\frac{2n+1}{(2n-1)(n+1)^2}\right].$$
Indications : dans ces produits le calcul formel n'est utile que parce qu'il permet d'expérimenter facilement. Pour le premier produit, regarder le développement du logarithme des termes ; pour le deuxième la décomposition en base 2 des produits tronqués ; pour le troisième, noter que, comme tout produit indéfini de fractions rationnelles, celui-ci s'exprime à l'aide de la fonction $\Gamma$ ; il ne reste plus ensuite qu'à faire tendre $n$ vers l'infini.
6. Nature des produits
$$\prod_{n>1}(1-1/n)^x(1+x/n),\quad \prod_{n\geq 0}\frac{x+x^{2n}}{1+x^{2n}}.$$
7. Nature des séries
$$\sum_{n\geq 0}\log(\cos\frac{x}{2^n}),\quad \sum_{n\geq 2}\frac{e^{inx}}{n\log n},\quad \sum_{n\geq 1}\frac{\cos(\log n)}{n}.$$
Pour la troisième, on pourra encadrer la valeur absolue des blocs de signe constant.
8. Étudier les domaines de convergence uniforme de la série
$$\sum_{n\geq 1}\frac{\sin nx}{\sqrt{n}},\qquad x\in[0,\pi].$$

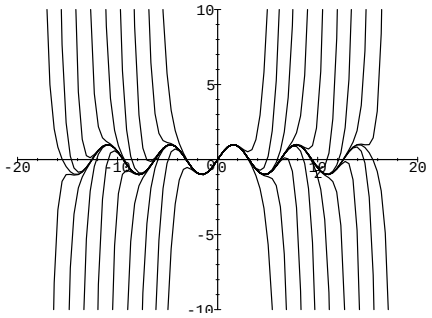



FIGURE 3 $\sin(z)$.

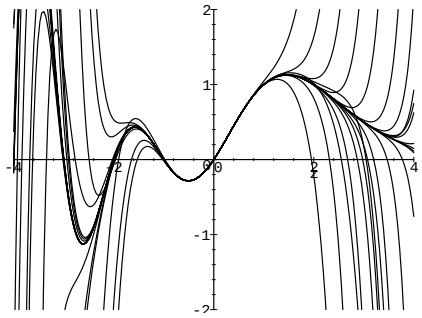



FIGURE 4 $1/\Gamma(z)$.

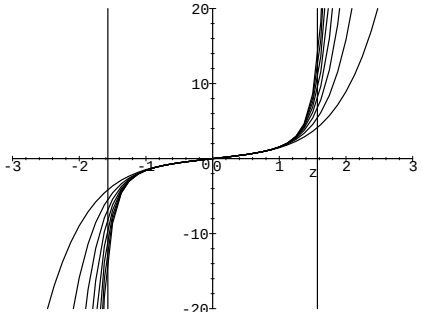



FIGURE 5 $\tan(z)$.

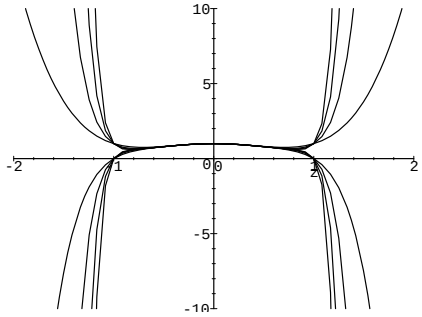



FIGURE 6 $\dfrac{1}{1+z^2}$.

## 2. Séries entières et développements limités

Les développements limités sont des objets de base du calcul formel. Maple leur affecte un type de données particulier (le type `series`) et de nombreuses opérations peuvent être effectuées sur des objets de ce type. Par ailleurs, lorsque le terme général d'une série entière est connu explicitement, quelques calculs sont possibles mais peu d'opérations symboliques peuvent être alors appliquées directement sur la série.

**2.1. Disque de convergence.** Le premier objet d'un développement en série est d'approximer la fonction localement. Les figures 3 à 6 montrent comment les développements à des ordres de plus en plus élevés s'approchent de la fonction. Ces figures ont été obtenues par l'instruction

```
plot({seq(convert(series(f,z,i),polynom),i=1..p)},z=a..b,c..d)
```

pour des valeurs appropriées de `p`, `a`, `b`, `c` et `d`. Les deux premières fonctions ont un développement en série entière dont le rayon de convergence est infini. Les deux autres ont un rayon de convergence fini, le premier à cause de singularités réelles et le second à cause de singularités imaginaires.

Au-delà du disque de convergence, la série cesse de converger, ce qui apparaît bien sur ces figures. Lorsque les coefficients $u_n$ d'une série entière sont connus explicitement, il n'est généralement pas trop difficile d'en déduire le rayon de convergence par la règle d'HADAMARD : c'est l'inverse de la limite supérieure de la suite $|u_n|^{1/n}$ si elle existe. Dans la plupart des cas en pratique, la limite supérieure est en fait la limite et la fonction `limit` peut la trouver.

**2.2. Fonctions définies explicitement.** C'est la commande `series` qui permet de calculer des développements limités. Voici comment on calcule le développement à l'ordre[†] 2 en 0 de

$$\left[\sin\left(\frac{\pi}{6+x}\right)+\cos\left(\frac{\pi}{3+x}\right)\right]^{1/x}.$$

```
series((sin(Pi/(6+x))+cos(Pi/(3+x)))^(1/x),x,4);
```

$$e^{\sqrt{3}\pi/24}+e^{\sqrt{3}\pi/24}\left(-\frac{61}{10368}\pi^2-\frac{7\sqrt{3}}{432}\pi\right)x$$
$$+e^{\sqrt{3}\pi/24}\left(\frac{5\sqrt{3}}{864}\pi+\frac{565}{124416}\pi^2+\frac{859\sqrt{3}}{4478976}\pi^3+\frac{3721}{214990848}\pi^4\right)x^2+O\left(x^3\right)$$

Cette commande ne se limite d'ailleurs pas à des développements en série entière, mais calcule aussi des développements en série de LAURENT (où des puissances négatives interviennent) et certains développements en série de PUISEUX (où des puissances rationnelles interviennent) :

```
series(arccos(1-x),x);
```

$$\sqrt{2}x^{1/2}+\frac{\sqrt{2}}{12}x^{3/2}+\frac{3\sqrt{2}}{160}x^{5/2}+\frac{5\sqrt{2}}{896}x^{7/2}+\frac{35\sqrt{2}}{18432}x^{9/2}+\frac{63\sqrt{2}}{90112}x^{11/2}+O(x^6)$$

La commande `series` peut parfois être améliorée en efficacité, notamment sur les développements avec un grand nombre de termes (voir le §2.2.4 du chapitre IV pour le cas des fractions rationnelles).

**2.3. Fonctions définies implicitement.** Le plus souvent, ce n'est pas de fonctions connues explicitement que l'on cherche un développement, mais plutôt d'une fonction définie implicitement comme solution d'une équation. Le développement permet alors de calculer des valeurs approchées à faible coût, en plus de donner une idée du comportement local. Nous montrons ici comment rechercher une série par coefficients indéterminés, puis nous présentons l'inversion de séries formelles et quelques-unes de ses applications.

On considère le problème de CAUCHY :

$$y''(z)-\sin(z)y'(z)=e^{\sin(y(z^2))},\qquad y(0)=0,y'(0)=1.$$

[†]La relation entre le dernier argument de `series` et l'ordre que l'on souhaite obtenir n'est pas aussi claire que désirable : l'argument donné à `series` est en gros l'ordre utilisé dans les calculs intermédiaires. Lorsqu'il y a beaucoup d'annulations, on obtient donc moins de termes qu'on le voudrait. La valeur par défaut de ce nombre de termes est gouvernée par la variable globale `Order`.

La méthode la plus directe consiste à substituer une série arbitraire dans cette équation, puis à effectuer le développement en série de l'expression obtenue et enfin à résoudre le système formé de l'ensemble des coefficients. En Maple, ceci s'exprime ainsi :

```
eq:=diff(y(z),z,z)-sin(z)*diff(y(z),z)-exp(sin(y(z^2))):
f:=z->z+convert([seq(a[i]*z^i,i=2..9)],`+`):
s:=series(eval(subs(y=f,eq)),z,8):
subs(solve({seq(coeff(s,z,i),i=0..7)},
    {seq(a[i],i=2..9)}),f(z));
```

$$z+\frac{z^2}{2}+\frac{z^3}{6}+\frac{z^4}{6}+\frac{z^5}{60}+\frac{z^6}{20}+\frac{z^7}{5040}+\frac{311}{20160}z^8-\frac{43}{362880}z^9$$

Cette méthode est dans certains cas inapplicable parce que l'étape du calcul de la série après substitution d'un polynôme de degré élevé dans l'équation peut être très exigeante en mémoire. Un simple procédé itératif l'emportera pratiquement toujours en place mémoire et souvent en vitesse :

```
for i from 2 to 9 do
   sol:=subs(ii=i,
       proc(z)z+convert([seq(a[j]*z^j,j=2..ii)],`+`)end);
   a[i]:=solve(op(1,series(eval(subs(y=sol,eq)),z,i)),a[i])
od: sol(z);
```

Un cas important qui ne nécessite pas le calcul par coefficients indéterminés est l'inversion de séries formelles. Il s'agit de la résolution de l'équation $x = f(y)$, où $f$ est connue par son développement en série. Maple dispose d'une fonction spécialisée pour cela et il faut essayer de s'y ramener aussi souvent que possible.

Cette fonction apparaît en Maple comme un avatar de la commande `solve`, mais elle réclame des conditions très particulières : il faut que l'un des membres de l'équation soit une série et que l'on résolve par rapport à la variable de la série. En outre, la commande fonctionne "mieux" (c'est-à-dire sans oublier de solution) lorsque la série comporte une partie linéaire et lorsque l'autre membre de l'équation est réduit à une variable.

Par exemple, le développement de $y(x)$ défini par

$$\frac{\log(1+y)}{y}=1+\sin(x),$$

est obtenu par

```
solve(series(log(1+y)/y,y)=1+sin(x),y);
```

$$-2x+\frac{8}{3}x^2-\frac{25}{9}x^3+\frac{344}{135}x^4+O\left(x^5\right)$$

Si l'inversion n'avait pas fonctionné, alors il aurait fallu aider le système ainsi :

```
series(subs(u=sin(x),solve(series(log(1+y)/y-1,y)=u,y)),x);
```

Lorsque la série ne comporte pas de terme linéaire, il vaut mieux en imposer un par changement de variable et rechanger la variable après l'inversion.

Par exemple l'équation

$$\frac{\sin y}{y} = \cos(x) + x^3,$$

se résout en posant $u = \sqrt{1 - \cos x - x^3}$ :

```
series(subs(u=(1-cos(x)-x^3)^(1/2),
   solve(series((1-sin(y)/y)^(1/2),y)=u,y)),x);
```

$$\frac{\sqrt{6}\sqrt{2}}{2}x - \frac{\sqrt{6}\sqrt{2}}{2}x^2 - \frac{7}{30}\sqrt{6}\sqrt{2}x^3 + O\left(x^4\right)$$

Une des applications de cette technique est la recherche du développement asymptotique des solutions de certaines équations. L'exemple classique est l'étude de la $n^{\text{e}}$ racine de l'équation $x = \tan x$. D'après la figure 1 p. 173, les courbes $y = \tan x$ et $y = x$ ont une intersection dans chacun des intervalles $](2n-1)\pi/2, (2n+1)\pi/2[$, et l'abscisse de cette intersection se rapproche de $(2n+1)\pi/2$ quand $n$ croît. Donc

$$x_n = (2n+1)\pi/2 - u_n = \tan(x_n).$$

Pour ramener ce problème à l'inversion de série, il faut utiliser la périodicité de la tangente et ramener toutes les séries au voisinage de 0. On obtient ainsi le développement de $x_n$ :

```
asympt(subs(t=2/Pi/(2*n+1),1/t-
    solve(t=series(1/(u+tan(Pi/2-u)),u),u)),n);
```

$$\pi n + \frac{\pi}{2} - \frac{1}{\pi n} + \frac{1}{2\pi n^2} - \left(\frac{1}{4\pi} + \frac{2}{3\pi^3}\right)\frac{1}{n^3} + \left(\frac{1}{8\pi} + \frac{1}{\pi^3}\right)\frac{1}{n^4} - \left(\frac{1}{16\pi} + \frac{1}{\pi^3} + \frac{13}{15\pi^5}\right)\frac{1}{n^5} + O\left(\frac{1}{n^6}\right).$$

**2.4. Sommes et produits.** Du point de vue du calcul formel, la détermination du développement en série d'une somme se réduit souvent à sommer les développements des sommants, la justification de la convergence devant être faite séparément.

Ainsi, le développement en série entière de

$$\sum_{n\geq 0} \sin(a^n x), \qquad a \in \,]-1, 1[\,,$$

est calculé par

```
series(sum(sin(a^n*x),n=0..infinity),x);
```

$$-\frac{x}{a-1} + \frac{x^3}{6\,(a^3-1)} - \frac{x^5}{120\,(a^5-1)} + O\left(x^6\right)$$

Il est alors facile de donner le terme général de la série (mais aucun système de calcul formel ne dispose de fonctionnalité pour ce faire).

De la même manière, le logarithme du produit

$$\prod_{n=0}^{\infty}\left(1 - \frac{z}{2^n}\right),$$

est une somme que Maple sait développer. Le résultat s'en déduit en prenant l'exponentielle :

```
series(exp(sum(log(1-z/2^n),n=0..infinity)),z);
```

$$1 - 2z + \frac{4}{3}z^2 - \frac{8}{21}z^3 + \frac{16}{315}z^4 - \frac{32}{9765}z^5 + O\big(z^6\big)$$

Là encore il est facile de repérer le motif général (regarder le quotient de deux dénominateurs consécutifs) et dans ce cas encore, il n'est pas difficile de prouver que la formule suggérée par le calcul est exacte.

Tous les cas ne sont pas aussi simples ; voici un exemple plus délicat où on cherche un équivalent lorsque $x \to 1^-$ de

$$\sum_{n\geq 0} \frac{x^n}{1+x^n}.$$

Lorsque $x \to 1^-$, tous les termes tendent vers $1/2$ et la fonction `series` ne peut pas être appliquée directement. On commence donc par récrire la somme :

```
simplify(subs(u=x^n,series(subs(x^n=u,x^n/(1+x^n)),u))):
map(sum,",n=0..infinity);
```

$$-\frac{1}{x-1} + \frac{1}{x^2-1} - \frac{1}{x^3-1} + \frac{1}{x^4-1} - \frac{1}{x^5-1} + \sum_{n=0}^{\infty} O(x^{6n})$$

Ce calcul suggère l'égalité suivante qu'il faudrait par ailleurs prouver :

$$\sum_{n\geq 0} \frac{x^n}{1+x^n} = \sum_{n>0} \frac{(-1)^n}{x^n-1}.$$

Tous les termes de la somme de droite ont un pôle simple en 1, on va donc intervertir (ceci reste à justifier) la limite et la somme :

```
sum(limit((-1)^n*(1-x)/(x^n-1),x=1),n=1..infinity)/(1-x);
```

$$\frac{\ln 2}{1-x}$$

**2.5. Intégrales.** Pour les intégrales comme pour les sommes, tant que l'on se contente de calculer sans chercher à justifier les interversions des signes sommes et intégrales, il n'y a pas de difficulté importante.

Ainsi le calcul du développement d'une intégrale comme

$$\int_x^{x^2} \frac{dt}{\sqrt{1+t^4}}$$

en $x = 0$ est facile, en admettant sa validité[†] :

```
series(Int(1/sqrt(1+t^4),t=x..x^2),x);
```

$$-x + x^2 + \frac{x^5}{10} + O\big(x^9\big)$$

[†]On utilise ici `Int` et non `int` pour éviter que Maple ne passe du temps à essayer de calculer symboliquement l'intégrale.

EXEMPLE 8. Lorsque $x$ tend vers 0, l'intégrale

$$\int_0^{+\infty} \frac{\cos t}{(1+t)^x}\, dt$$

devient difficile à évaluer par les méthodes de calcul numérique d'intégrales. Son développement en série devient alors préférable. Il ne suffit cependant pas pour l'obtenir d'intégrer terme à terme le développement de l'intégrande, car alors l'intégrale du terme constant est divergente. Le résultat s'obtient en commençant par une intégration par parties :

```
student[intparts](Int(cos(t)/(1+t)^x,t=0..A),1/(1+t)^x);
```

$$\frac{\sin(A)}{(1+A)^x} + \int_0^A \frac{x\sin(t)}{(1+t)^x\,(1+t)} dt$$

Lorsque $A$ tend vers l'infini, le premier terme tend vers 0 pour tout $x > 0$ et le second se traite directement :

```
series(subs(A=infinity,op(2,")),x,4);
```

$$\left[\frac{\pi\cos(1)}{2} - \mathrm{Si}(1)\cos(1) + \mathrm{Ci}(1)\sin(1)\right] x - \int_0^\infty \frac{\ln(1+t)\sin(t)}{1+t} dt\, x^2 + \int_0^\infty \frac{\ln(1+t)^2\sin(t)}{2+2t} dt\, x^3 + O(x^4)$$

Dans cette expression, Ci et Si désignent le cosinus intégral et le sinus intégral. On peut alors calculer une fois pour toutes les valeurs numériques de ces coefficients de TAYLOR et utiliser ce développement pour calculer des valeurs de l'intégrale de départ.

Nous verrons des exemples plus délicats de calculs de développements d'intégrales en §3.5.

**2.6. Séries génératrices.** À plusieurs reprises dans ce livre, nous avons vu qu'il était difficile, voire impossible, de faire comprendre à Maple que telle variable représentait un réel positif, un entier impair,... Bien souvent, les informations liées aux variables représentant des entiers peuvent être prises en compte par le biais de *séries génératrices* (voir chap. VII). Nous insistons ici sur l'utilisation de séries génératrices pour le calcul exact d'expressions faisant intervenir un paramètre entier. Dans la section suivante, nous reviendrons sur les séries génératrices pour leur application à la recherche du développement asymptotique d'une suite.

Une illustration typique de l'usage des séries génératrices pour calculer des expressions dépendant d'un entier est le calcul d'une série de FOURIER. Les premiers coefficients sont calculés aisément par la commande `int`, mais il est parfois possible de calculer explicitement le $n^{\mathrm{e}}$ coefficient :

```
f:=1/(lambda^2-2*lambda*cos(x)+1);
```

$$f := \frac{1}{\lambda^2 - 2\lambda\cos(x) + 1}$$

Comme $f$ est paire en $x$, les coefficients à calculer sont :

$$\int_0^{2\pi} f(x)\cos(nx)\,dx.$$

Le calcul commence par la détermination de la série génératrice de $\cos(nx)$ :

```
g:=evalc(Re(sum(exp(I*x)^n*z^n,n=0..infinity)));
```

$$g := \frac{1 - z\cos(x)}{(1 - z\cos(x))^2 + \sin(x)^2 z^2}$$

Il ne reste plus qu'à intervertir (ce qu'il faudrait justifier) les signes somme et intégrale et à simplifier le résultat (voir chap. I) :

```
res:=int(f*g,x=0..2*Pi):
assume(z<1,z>-1):assume(lambda,real):
simplify(factor(res),radical);
```

$$\begin{aligned}
&\operatorname{signum}(-1+\lambda^{\tilde{}2})\operatorname{signum}(-1+z^{\tilde{}2})\left(-z^{\tilde{}}\operatorname{signum}(-1+\lambda^{\tilde{}2})\right.\\
&\quad + z^{\tilde{}}\operatorname{signum}(-1+\lambda^{\tilde{}2})\lambda^{\tilde{}2} - \%_1 z^{\tilde{}}\lambda^{\tilde{}2}\operatorname{signum}(-1+z^{\tilde{}2})\\
&\quad \left.-\%_1 z^{\tilde{}}\operatorname{signum}(-1+z^{\tilde{}2}) + 2\%_1\lambda^{\tilde{}}\operatorname{signum}(-1+z^{\tilde{}2})\right)\pi\\
&\quad /\left((-1+\lambda^{\tilde{}2})(\lambda^{\tilde{}}z^{\tilde{}}-1)(-\lambda^{\tilde{}}+z^{\tilde{}})\right)\\
&\quad \%_1 := \operatorname{csgn}\left((\lambda^{\tilde{}}+1)^2\operatorname{signum}(-1+\lambda^{\tilde{}2})(-1+\lambda^{\tilde{}2})\right)
\end{aligned}$$

```
subs(csgn=1,signum(z^2-1)=-1,"):
normal(map(subs,[seq(signum(lambda^2-1)=i,i=[-1,1])],"));
```

$$\left[\frac{2\pi}{(1-\lambda^{\tilde{}}z^{\tilde{}})(1-\lambda^{\tilde{}2})}, \frac{2\pi\lambda^{\tilde{}}}{(\lambda^{\tilde{}}-z^{\tilde{}})(\lambda^{\tilde{}2}-1)}\right]$$

Le résultat obtenu dépend du signe de $\lambda^2 - 1$. Comme il s'agit dans les deux cas de fractions rationnelles très simples, le coefficient de $z^n$ s'en déduit aisément :

$$\int_0^{2\pi} \frac{\cos(nx)\,dx}{\lambda^2 - 2\lambda\cos(x) + 1} = \frac{2\pi}{|\lambda^2-1|}\cdot\begin{cases}\lambda^n & \text{si } \lambda^2 < 1,\\ \lambda^{-n} & \text{si } \lambda^2 > 1.\end{cases}$$

### 2.7. Exercices.

1. Déterminer un développement limité solution de l'équation
$$y'(z) - y(y(z)) = \cos(z), \qquad y(0) = 0,$$
d'abord à l'ordre 6, puis à l'ordre 50.
2. Calculer le développement asymptotique lorsque $n$ tend vers l'infini de la $n^{\text{e}}$ racine de $\tan x = x^3/(x^2-1)$.
3. Dans l'exemple du développement en série entière du sinus intégral de Fresnel du §1.1, calculer le développement en série de l'indice du terme maximal.
4. Déterminer un équivalent lorsque $x$ tend vers $1^-$ de
$$\sum_{n\geq 1}\frac{nx^n}{1-x^n}.$$

5. Calculer le développement en série entière de
$$\int_0^{\pi/2} \log(1 + x \sin^2 t)\, dt.$$
Vérifier que pour $x = 1/10$ on obtient une très bonne approximation.
6. Calculer les premiers termes du développement en puissances de $1/n$ de l'intégrale
$$\int_0^{+\infty} \sin(x^n)\, dx$$
lorsque $n \to \infty$. [Indication : commencer par changer la variable.]
7. Calculer le développement en série entière en $r$ de
$$F(x, r) = \frac{1}{2\pi} \int_0^{2\pi} \frac{(1 - r^2) f(t)}{1 - 2r \cos(t - x) + r^2}\, dt.$$
[On regardera avec profit le `help` de `combine/trig`.]
8. Calculer
$$\int_\alpha^{\alpha+2\pi} \frac{\sin^2 n\frac{x-t}{2}}{\sin^2 \frac{x-t}{2}} \frac{dt}{n}, \qquad n \in \mathbb{N}^\star.$$
9. Calcul de la fonction perturbatrice de la Lune (Delaunay). La fonction perturbatrice de la Lune est à un facteur multiplicatif près
$$S = \frac{k'^3}{k^2} P_2(\cos\theta) + c\frac{k'^4}{k^3} P_3(\cos\theta) + c^2 \frac{k'^5}{k^4} P_4(\cos\theta) + \cdots \tag{3}$$
où $k$ (respectivement $k'$) est le rapport de la distance moyenne de la Terre à la Lune (respectivement au Soleil) à la distance à l'instant considéré, les $P_i$ sont les polynômes de Legendre, et où $\theta$ est l'angle entre les directions Terre-Lune et Terre-Soleil. La première étape du calcul consiste à obtenir un développement de $k$, $k'$ et $\cos\theta$. Cette dernière quantité est liée à une constante $\gamma$ et aux angles $f, f'$ de rotation de la Lune et du Soleil par l'identité
$$\cos\theta = (1 - \gamma^2)\cos(f + h - f' - h') + \gamma^2 \cos(f + h + f' + h'), \tag{4}$$
où $h, h'$ sont d'autres angles indépendants. Le rapport $k$ est donné par
$$k = \frac{db}{dl} \tag{5}$$
où $b$ est donné par l'équation de Kepler
$$b = l + e \sin b \tag{6}$$
et finalement $f$ est donné par
$$\frac{df}{dl} = k^2 \sqrt{1 - e^2}. \tag{7}$$
Le but de l'exercice est de calculer un développement de la fonction perturbatrice à l'ordre $n$ (commencer par $n = 2$) en fonction des petites quantités $e, e', \gamma$ et $c$. Les fonctions Maple à utiliser sont indiquées entre crochets.

(1) À l'aide de l'équation de KEPLER (6), calculer un développement de $b$ à l'ordre $n$ en $e$. [Inversion de séries formelles.]
(2) À partir du développement de $b$, en déduire celui de $k$ par l'équation (5) et celui de $f$ par (7). Remplacer $e$ par $e'$ et $l$ par $l'$ dans $k$ et $f$ pour obtenir les développements de $k'$ et $f'$ relatifs au Soleil.
(3) Calculer $\cos\theta$ en fonction de $\gamma, e, e', l, l', h, h'$ à l'aide de l'identité (4).
(4) Former $S$ en se servant de la formule (3). Il suffit de considérer les polynômes de LEGENDRE jusqu'à l'ordre $\lfloor n/2 \rfloor + 2$. [`orthopoly[P]`]
(5) Effectuer un développement limité de $S$ à l'ordre $n$ en les quatre variables $e, e', \gamma, c$. Étant données les valeurs approchées $e \simeq 1/18$, $e' \simeq 1/60$, $\gamma \simeq 1/11$ et $c \simeq 1/400$, on considère comme DELAUNAY $e$ et $\gamma$ comme des termes d'ordre un, ${e'}^3$ comme un terme d'ordre quatre et $c$ comme un terme d'ordre deux. [`mtaylor,select`]
(6) Linéariser les expressions trigonométriques afin de mettre $S$ sous la forme d'une somme de termes du genre $a_i \cos b_i$ où $a_i$ est un monôme en $e, e', \gamma, c$ et $b_i$ est une combinaison linéaire des angles $l, l', h, h'$. [`combine(···,trig)`]
(7) Regrouper les termes ayant le même facteur $\cos b_i$. [Commandes `collect` et `indets`.]

On doit obtenir à l'ordre 2 le développement suivant.

$$\begin{aligned}
S = {} & \frac{1}{4} - \frac{3}{2}\gamma^2 + \frac{3}{8}e^2 + \frac{3}{8}c\cos(l - l' + h - h') - \frac{1}{2}e\cos(l) + \frac{3}{4}e'\cos(l') \\
& + \left(-\frac{3}{2}\gamma^2 + \frac{3}{4} - \frac{15}{8}e^2\right)\cos(2l - 2l' + 2h - 2h') - \frac{1}{8}e^2\cos(2l) \\
& + \frac{3}{4}e\cos(3l - 2l' + 2h - 2h') + \frac{21}{8}e'\cos(2l - 3l' + 2h - 2h') \\
& - \frac{3}{8}e'\cos(2l - l' + 2h - 2h') + \frac{3}{4}e^2\cos(4l - 2l' + 2h - 2h') \\
& + \frac{5}{8}c\cos(3l - 3l' + 3h - 3h') - \frac{9}{4}e\cos(l - 2l' + 2h - 2h') \\
& + \frac{15}{8}e^2\cos(-2l' + 2h - 2h') + \frac{3}{2}\gamma^2\cos(2l' + 2h') + \frac{3}{2}\gamma^2\cos(2l + 2h).
\end{aligned}$$

## 3. Développements asymptotiques

Tous les comportements asymptotiques ne s'expriment pas en termes de séries entières. Il faut souvent faire intervenir des singularités logarithmiques ou exponentielles. Du point de vue du calcul formel cela signifie une structure de données et des opérations différentes. À l'heure actuelle, les systèmes de calcul formel sont encore assez démunis pour manipuler des développements asymptotiques dans des échelles asymptotiques très fines.

La commande de base pour les développements asymptotiques en Maple est `asympt`, mais il faut occasionnellement se servir de `series`, ou de fonctions plus spécifiques, comme `eulermac`, que nous verrons dans cette section.

Le calcul asymptotique est un art subtil et on ne peut pas donner de méthode générale infaillible. Cependant, il existe dans ce domaine un certain nombre de méthodes souvent applicables et dont la complexité peut être extrêmement amoindrie par l'usage d'un système de calcul formel. Nous présentons dans cette section certaines de ces méthodes.

**3.1. Fonctions définies explicitement.** Les calculs de développements asymptotiques de fonctions connues explicitement sont importants car ils forment la base de nombreuses opérations courantes : calcul numérique, comparaison d'ordres de grandeurs de fonctions, calcul de développements asymptotiques plus complexes, étude de la convergence d'une intégrale impropre,...

La commande de base en Maple est `asympt` ; voici comment faire apparaître la formule de STIRLING :

```
asympt(n!,n,4);
```

$$\left(\frac{n^n}{e^n}\right)\left[\sqrt{2}\sqrt{\pi}\sqrt{n}+\frac{\sqrt{2}\sqrt{\pi}}{12\sqrt{n}}+\frac{\sqrt{2}\sqrt{\pi}}{288n^{3/2}}-\frac{139\sqrt{2}\sqrt{\pi}}{51840n^{5/2}}+O\left(n^{-7/2}\right)\right]$$

Cependant la commande `asympt` est encore très insuffisante. Il est en effet difficile d'obtenir des développements simples comme celui de

$$\binom{n}{\frac{2}{3}n+p}^2 \Big/ \binom{n}{\frac{2}{3}n}^2$$

lorsque $n \to \infty$. Voici les acrobaties par lesquelles on peut y parvenir :

```
f:=binomial(n,2*n/3+p)^2/binomial(n,2*n/3)^2:
expand(ln(convert(f,GAMMA)),GAMMA);
```

$$-2\ln\Gamma(2n/3+p+1)-2\ln\Gamma(n/3-p+1)+2\ln\Gamma(2n/3+1)+2\ln\Gamma(n/3+1)$$

```
eval(subs(GAMMA=(x->exp(_B)),_B=asympt(ln(GAMMA(x)),x,4),")):
asympt(simplify(exp("),ln),n,3):
simplify(map(normal,"),exp);
```

$$2^{-2p}+\frac{2^{-2p}\left(3/2p-9/2p^2\right)}{n}+O\left(\frac{1}{n^2}\right)$$

**3.2. Fonctions définies implicitement.** La méthode des coefficients indéterminés qui s'appliquait au cas des développements limités ne s'applique plus ici, puisqu'il faut d'abord déterminer dans quelle échelle asymptotique doit se faire le développement. Nous montrons cependant sur un exemple d'inversion comment les calculs se mènent dans de nombreux cas.

EXEMPLE 9. On cherche le comportement asymptotique lorsque $x \to \infty$ de la fonction $y(x)$ définie par

$$ye^{\ln^2(y\exp\sqrt{\ln\ln y})} = x.$$

Une première remarque est que $y$ tend nécessairement vers l'infini avec $x$. Ensuite, la première étape est de comparer les deux facteurs du produit de gauche pour déterminer lequel domine. Il faut noter que Maple est le seul système de calcul formel à savoir faire cette comparaison :

```
eq:=y*exp(ln(y*exp(sqrt(ln(ln(y)))))^2)=x:
limit(y/op(2,op(1,eq)),y=infinity);
```

$$0$$

Comme c'est le second facteur qui domine, on divise les deux membres de l'égalité par le premier facteur et on inverse jusqu'au produit suivant :

```
assume(y>0,ln(y)>0,ln(ln(y))>0);
simplify(map(exp,map(t->t^(1/2),map(ln,map(t->t/y,eq)))));
```

$$\tilde{y} \exp\sqrt{\ln\ln \tilde{y}} = \exp\sqrt{\ln x - \ln \tilde{y}}$$

À nouveau, la comparaison des deux facteurs du produit de gauche montre que le premier l'emporte, donc on divise par le second :

```
op(1,")/op(2,op(1,"))=op(2,")/op(2,op(1,"));
```

$$\tilde{y} = \frac{\exp\sqrt{\ln x - \ln \tilde{y}}}{\exp\sqrt{\ln\ln \tilde{y}}}$$

On voudrait raffiner l'estimation grossière $\exp(\sqrt{\ln x})$ que nous suggère cette équation. Si `asympt` était bien fait, il suffirait de taper

```
t:=op(2,"):
exp(sqrt(ln(x))):
while "<>"" do asympt(subs(y=",t),x) od: ";
```

Malheureusement, Maple ne permet pas de faire cela directement. Voici les commandes qui amènent péniblement au résultat :

```
t:=subs(ln(x)=X,simplify(log(op(2,")),ln)):
X^(1/2):
while "<>"" do
  map(simplify,asympt(subs(ln(y)=",t),X,1)) od:
subs(X=ln(x),exp(op(1,")+op(2,")+op(3,"))*simplify(
  map(normal,asympt(exp(subsop(1=0,2=0,3=0,")),X,1)),ln));
```

$$e^{\sqrt{\ln x}-\frac{\sqrt{2}}{2}\sqrt{\ln\ln x}-\frac{1}{2}}\left[1+\frac{\sqrt{2}\sqrt{\ln\ln x}+\frac{5}{2}\ln\ln x+\sqrt{2}(\ln\ln x)^{3/2}}{4\ln\ln x\sqrt{\ln x}}+O\Big(\frac{1}{\ln x}\Big)\right]$$

(Le numérateur du $O$ renvoyé par Maple peut sembler faux par excès d'optimisme, mais il est dû à une étrange définition du $O$ en Maple, décrite dans la documentation de `series`.)

**3.3. Sommes.** Une méthode importante pour déterminer le comportement asymptotique d'une somme de termes assez réguliers est de comparer la somme avec une intégrale. Nous donnons ici deux exemples de calcul asymptotique et le calcul peut se pousser jusqu'à obtenir un encadrement de la somme.

Exemple 10. On cherche la limite de

$$n^3 \sum_{k=1}^{n} \frac{1}{n^4+n^2k^2+k^4}.$$

Cette suite se récrit $1/n \sum f(k/n)$, où $f = 1/(1+x^2+x^4)$. Asymptotiquement, la somme tend donc vers l'intégrale

```
int(1/(1+x^2+x^4),x=0..1);
```

$$\frac{\sqrt{3}\pi}{12} + \frac{\ln(3)}{4}$$

Sans entrer dans les détails, disons que cette méthode de comparaison d'une somme avec une intégrale (la formule d'EULER-MACLAURIN ci-dessous en est un cas particulier) demande un peu d'efforts pour l'obtention d'une majoration des termes d'erreur : il faut une majoration uniforme sur les dérivées de la fonction intégrée.

Lorsque les sommants $u_k$ ne dépendent pas de la borne de sommation $n$, la formule d'EULER-MACLAURIN s'applique. Elle renvoie une expression $S_k$ telle qu'asymptotiquement $S_{k+1} - S_k \approx u_k$. Pour trouver le comportement asymptotique de

$$\sum_{k=1}^{n-1} k^2 \log k$$

lorsque $n \to \infty$, la commande `eulermac` s'utilise ainsi :

```
readlib(eulermac)(n^2*log(n),n);
```

$$\frac{n^3\ln(n)}{3} - \frac{n^3}{9} - \frac{n^2\ln(n)}{2} + \frac{n\ln(n)}{6} + \frac{n}{12} - \frac{1}{360n} + \frac{1}{7560n^3} + O\left(\frac{1}{n^5}\right)$$

Comme `int`, la commande `eulermac` sait aussi traiter des sommes définies, et fournit alors une constante d'intégration qui n'apparaît pas dans le cas indéfini :

```
eulermac(k^2*log(k),k=1..n);
```

$$\frac{\ln(n)n^3}{3} - \frac{n^3}{9} - \zeta(1,-2) + \frac{\ln(n)n^2}{2} + \frac{\ln(n)n}{6} + \frac{n}{12} - \frac{1}{360n} + \frac{1}{7560n^3} + O\left(\frac{1}{n^5}\right)$$

La constante $\zeta(1,-2)$ représente la dérivée de la fonction zeta de RIEMANN en $-2$, et vaut $\zeta'(-2) \approx -0.03$.

```
evalf(subs(n=100,")=convert([seq(k^2*log(k),k=1..100)],`+`));
```

$$1447056.584 + O(1/10000000000) = 1447056.585$$

La commande `asympt` sait reconnaître les cas d'application de la méthode d'EULER-MACLAURIN ; on aurait donc pu écrire directement

```
asympt(sum(k^2*log(k),k=1..n),n);
```

La comparaison avec une intégrale ne fonctionne bien que lorsque les sommants sont assez réguliers et que la somme n'est pas trop concentrée : si seuls quelques termes contribuent asymptotiquement à la somme, il faut commencer par les repérer, puis majorer le reste. C'est le cas par exemple pour $\sum_{k=1}^{n} k!$.

**3.4. Suites itératives.** Nous reprenons l'exemple de la récurrence $u_{n+1} = \sin u_n$ du chapitre VII, pour en étudier la vitesse de convergence. La méthode que nous allons employer s'applique à toutes les suites convergeant vers 0 définies par $u_{n+1} = f(u_n)$, la fonction $f$ ayant un développement en série de la forme $f(x) = x(1 + \sum_{i\geq 1} a_i x^i)$ lorsque $x \to 0$. En notant $p$ le plus

petit indice tel que $a_p \neq 0$, l'idée est de regarder la suite $v_n = 1/u_n^p$. En substituant dans la récurrence on obtient

$$v_{n+1} - v_n = -pa_p + O(u_n). \tag{8}$$

Comme $u_n$ tend vers 0 et par hypothèse $a_p \neq 0$, à partir d'un certain indice la suite $\{v_{n+1} - v_n\}$ est de signe constant et $|v_{n+1} - v_n| > p|a_p|/2$. Donc pour $n$ suffisamment grand $|v_n| > p|a_p|n/4$ et par conséquent $u_n = O(n^{-1/p})$. Ensuite, il s'agit de raffiner l'estimation $v_n \sim -pa_p n$ en sommant (8). Pour cela, il faut prendre suffisamment de termes dans le développement (8) afin d'avoir un reste tendant vers 0 dans l'estimation de $v_n$. Ce résultat est atteint lorsque le reste est en $O(u_n^{p+\epsilon})$. La somme des $u_n^k$ pour $k > p$ converge alors vers une constante *qui dépend de* $u_0$ alors que tout le reste du développement n'en dépendait pas. Ce procédé se poursuit ensuite formellement sans connaître $u_0$.

Dans notre exemple, $p = 2$ et $a_p = -1/6$. Le calcul se déroule ainsi :

```
d:=series(1/sin(x)^2-1/x^2,x,7);
```

$$d := 1/3 + \frac{x^2}{15} + O\left(x^4\right)$$

Ici $d$ correspond à $v_{n+1} - v_n$. Une première approximation s'en déduit :

```
u:=(n/3)^(-1/2):
```

$v_n$ s'obtient alors en sommant $\{v_n - v_{n-1}\}$ :

```
v:=asympt(eulermac(asympt(subs(x=u,n=n-1,d),n),n),n);
```

$$v := \frac{n}{3} + \frac{\ln(n)}{5} - 1/6 + O\left(\frac{1}{n}\right)$$

La constante $-1/6$ obtenue par la méthode d'Euler-Maclaurin n'est valable qu'à une constante d'intégration près. Nous la remplaçons par une constante symbolique $C$ qui représente la valeur exacte.

```
v:=v+1/6+C:
u:=asympt(v^(-1/2),n);
```

$$u := \frac{\sqrt{3}}{\sqrt{n}} + \frac{\sqrt{3}\left(-3/10\ln n - 3/2C\right)}{n^{3/2}} + O\left(n^{-5/2}\right)$$

Pour obtenir plus de termes du développement, il suffit d'augmenter l'ordre des développements et d'appliquer répétitivement

```
Order:=Order+1: d:=subs(x=u,n=n-1,series(1/sin(x)^2-1/x^2,x)):
u:=asympt((C+1/6+eulermac(asympt(d,n),n))^(-1/2),n);
```

**3.5. Intégrales.** Plusieurs méthodes assez générales permettent de calculer le comportement asymptotique de nombreuses intégrales. L'intégration par parties vient à bout des intégrales les plus régulières. Sinon, la première partie de l'étude consiste à localiser les points du domaine d'intégration où l'intégrande se concentre lorsque le paramètre tend vers sa limite. L'intégrande est alors développée au voisinage de ces points, puis sommation et intégration sont intervertis. Il faut bien sûr justifier la validité de cette interversion. Lorsque l'intégrande est exponentiellement décroissante, cette justification peut prendre la forme d'un théorème assez général et la méthode s'appelle

alors la méthode de LAPLACE. Il est également possible d'utiliser des propriétés de la transformée de MELLIN pour déterminer le comportement asymptotique d'une intégrale, mais nous n'en parlerons pas ici.

EXEMPLE 11. Nous avons vu en §1.1 comment calculer numériquement le sinus intégral de FRESNEL lorsque l'argument est assez proche de l'origine. Lorsque l'argument devient plus grand, il est préférable d'utiliser le développement asymptotique. L'intégration par parties s'applique à l'intégrale de l'équation (1), après un changement de variables :

```
S := a -> Int(sin(Pi*t^2/2),t=a..infinity):
value(S(0))-student[changevar](t=u^(1/2),S(x),u);
```

$$1/2 - \int_{x^2}^{\infty} \frac{\sin(\frac{\pi u}{2})}{2\sqrt{u}} du$$

```
student[intparts](",u^(-1/2));
```

$$1/2 - \frac{\cos\left(\frac{\pi x^2}{2}\right)}{\pi x} + \int_{x^2}^{\infty} \frac{\cos(\frac{\pi u}{2})}{2u^{3/2}\pi} du$$

Le calcul se poursuit et donne de plus en plus de termes du développement :

```
for i from 3 by 2 to 7 do student[intparts](",u^(-i/2)) od:
subs(csgn(x)=1,map(simplify,"));
```

$$1/2 - \frac{\cos\frac{\pi x^2}{2}}{\pi x} - \frac{\sin\frac{\pi x^2}{2}}{\pi^2 x^3} + 3\frac{\cos\frac{\pi x^2}{2}}{\pi^3 x^5} + 15\frac{\sin\frac{\pi x^2}{2}}{\pi^4 x^7} - \int_{x^2}^{\infty} \frac{105\sin\frac{\pi u}{2}}{2\pi^4 u^{9/2}} du$$

EXEMPLE 12. On considère maintenant l'intégrale

$$\int_0^x \frac{\log(1+tx)}{1+t^2}\, dt,$$

où $x$ tend vers l'infini.

Le calcul se simplifie en remarquant que lorsque $x$ croît, $\log(1+tx)$ se comporte au premier ordre comme $\log(x)$, la différence étant égale à $\log(1/x+t)$. Une première partie du comportement asymptotique est donc donnée par

```
f1:=int(log(x)/(1+t^2),t=0..x);
```

$$f1 := \log(x)\arctan(x)$$

Il reste à considérer l'intégrale où $\log(1+tx)$ est remplacé par $\log(1/x+t)$. La commande `asympt` ne s'applique pas directement à cette intégrale parce qu'une des bornes dépend de $x$. Il faut donc encore décomposer en deux sous-problèmes : d'une part l'intégrale étendue jusqu'à l'infini, d'autre part l'intégrale de $x$ à $+\infty$. La première se traite directement par `asympt` :

```
f2:=asympt(Int(log(1/x+t)/(1+t^2),t=0..infinity),x);
```

$$f2 := \frac{\ln x + 1}{x} + \frac{\pi}{4x^2} + \frac{-\frac{\ln x}{3} - 1/9}{x^3} - \frac{\pi}{8x^4} + \frac{\frac{\ln x}{5} + 1/25}{x^5} + O\left(\frac{1}{x^6}\right)$$

Il nous reste l'intégrale

$$\int_x^{\infty} \frac{\ln(1/x+t)}{1+t^2} dx,$$

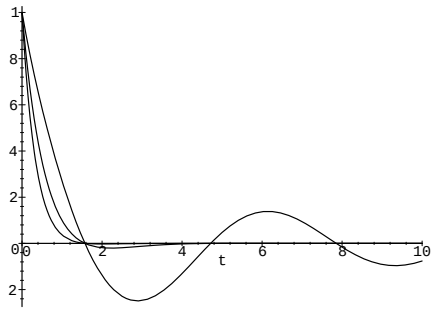



FIGURE 7
`plot({seq(exp(-lambda*t)*cos(t)/(1+t),lambda=[0,1,2])},t=0..10)`

pour laquelle il suffit d'intégrer terme à terme le développement de l'intégrande :

```
f3:=asympt(map(int,subs(O=0,asympt(ln(1/x+t)/(1+t^2),t)),
    t=x..infinity),x);
```

$$f3 := \frac{\ln x + 1}{x} + \frac{7/18 - \frac{\ln x}{3}}{x^3} + O\left(\frac{1}{x^5}\right)$$

Il ne reste plus qu'à sommer les trois contributions :

```
asympt(f1+f2-f3,x);
```

$$\frac{\pi \ln x}{2} - \frac{\ln x}{x} + \frac{\pi}{4x^2} + \frac{\frac{\ln x}{3} - 1/2}{x^3} - \frac{\pi}{8x^4} + O\left(\frac{1}{x^5}\right)$$

Toutes les étapes du calcul nécessitent justification. À défaut, une vérification numérique est réconfortante :

```
evalf(Int(ln(1+10*t)/(1+t^2),t=0..10)),evalf(subs(x=10,"));
```

$$3.394715756, \quad 3.394715938 + O(1/100000)$$

EXEMPLE 13. Dans sa version la plus simple, la méthode de LAPLACE s'applique à des intégrales du type

$$\int_0^\infty e^{-\lambda t} f(t)\, dt,$$

où $\lambda$ tend vers l'infini. Nous allons détailler le calcul avec

$$f(t) = \frac{\cos t}{1+t}.$$

L'idée est simple : quand $\lambda$ croît, l'intégrale se "concentre" au voisinage de l'origine (fig. 7). Il suffit alors de développer la partie indépendante de $\lambda$ au voisinage de l'origine et d'intégrer terme à terme :

```
series(cos(t)/(1+t),t):
readlib(laplace)(",t,lambda);
```

$$\frac{1}{\lambda} - \frac{1}{\lambda^2} + \frac{1}{\lambda^3} - \frac{3}{\lambda^4} + \frac{13}{\lambda^5} - \frac{65}{\lambda^6}$$

(la fonction `laplace` perd le $O$).

Bien souvent, l'intégrale ne se présente pas directement sous la forme ci-dessus et il faut l'y ramener par des changements de variables appropriés.

EXEMPLE 14. On étudie l'intégrale suivante lorsque $n$ tend vers l'infini :

$$\int_0^b \sin^n t\, dt, \qquad b \in ]0, \pi/2[\,.$$

Cette intégrale ne se concentre pas à l'origine, mais en $b$. Ceci suggère de récrire $\sin^n t$ comme $\exp(n \log \sin t)$ et d'effectuer le changement de variable $\log \sin t = \log \sin b - u$ :

```
f:=Int(sin(t)^n,t=0..b):
student[changevar](log(sin(t))=log(sin(b))-u,f,u):
simplify(",trig,exp);
```

$$\int_0^\infty \frac{(\sin(b)e^{-u})^n \sin(b)e^{-u}}{\sqrt{1-\sin(b)^2 e^{-2u}}} du$$

Il reste à isoler la partie qui dépend de $n$ de celle qui n'en dépend pas, à développer cette dernière et à intégrer terme à terme :

```
subs(csgn(cos(b))=1,sin(b)^(n+1)*laplace(simplify(
      series(exp(-u)/sqrt(1-sin(b)^2*exp(-2*u)),u,3)),u,n));
```

$$\sin(b)^{n+1}\left[\frac{1}{\cos(b)n} - \frac{1}{\cos(b)^3 n^2} - \frac{-3+2\cos(b)^2}{\cos(b)^5 n^3}\right]$$

Là encore, le terme en $O$ a été perdu, il faut penser à le réintroduire si l'on souhaite poursuivre le calcul.

**3.6. Solutions d'équations différentielles.** Les équations différentielles linéaires à coefficients polynomiaux jouent un rôle important en calcul formel : elles donnent accès par une information représentable en machine (les coefficients) à de nombreuses fonctions spéciales. L'une des opérations qui peut s'effectuer directement sur les équations, sans avoir à résoudre, est le calcul du comportement des solutions au voisinage d'un point du plan, même lorsqu'il s'agit d'une singularité. Les singularités se trouvent parmi les racines du coefficient de tête. Le comportement en leur voisinage est toujours une combinaison linéaire de développements de la forme :

$$\exp[Q(x^{-1/d})]x^\alpha \sum_{i=0}^{m} \sum_j c_{ij} x^{j/d} \ln^i x,$$

où $x = z - \rho$, $\rho$ est la singularité et $Q$ est un polynôme. Lorsque $Q = 0$ dans tous les développements, le point singulier est dit *régulier*. Sinon, il est *irrégulier*.

La théorie permet de calculer toutes les constantes intervenant dans ces expressions et de produire un développement complet. Maple ne connaît malheureusement que les points singuliers réguliers. Cette fonctionnalité est accessible par la commande `dsolve`, avec l'option `series` :

```
eq:=z^2*diff(y(z),z,z)-z*diff(y(z),z)-(z+1)*y(z)=0;
```

$$z^2\frac{\partial^2}{\partial z^2}y(z) - z\frac{\partial}{\partial z}y(z) - (z+1)y(z) = 0$$

```
Order:=3:
simplify(dsolve(eq,y(z),series));
```

$$y(z) = \_C_1 z^{1-\sqrt{2}}\left[1 - \frac{z}{-1+2\sqrt{2}} + \frac{z^2}{4\left(-1+2\sqrt{2}\right)\left(\sqrt{2}-1\right)} + O\left(z^3\right)\right] + \_C_2 z^{1+\sqrt{2}}\left[1 + \frac{z}{1+2\sqrt{2}} + \frac{z^2}{4\left(1+2\sqrt{2}\right)\left(1+\sqrt{2}\right)} + O\left(z^3\right)\right]$$

Lorsque la singularité n'est pas à l'origine, il faut d'abord l'y ramener par un changement de variables.

**3.7. Séries génératrices.** Le comportement asymptotique d'une suite se lit sur les singularités de sa fonction génératrice. La position des singularités de plus petit module donne la croissance exponentielle. Le comportement de la fonction au voisinage de ces singularités donne la croissance sous-exponentielle. Un exemple de cette méthode de calcul asymptotique a déjà été présenté, pour le cas des suites solutions de récurrences linéaires à coefficients constants. Voici quelques autres exemples pour illustrer cette méthode. Dans tous les cas les singularités sont polaires ou algébriques isolées, mais la méthode s'étend à d'autres types de singularités.

Formellement, les opérations à faire sont les suivantes :

- localiser la ou les singularités de module minimal ;
- calculer le développement de la fonction génératrice au voisinage de cette (ces) singularité(s) ;
- prendre les coefficients terme à terme, en utilisant le fait que

$$(1-z/\rho)^\alpha = \sum_{n=0}^{\infty} \frac{\Gamma(n-\alpha)}{\Gamma(-\alpha)}\frac{z^n}{n!\rho^n}, \qquad \alpha \notin \mathbb{N}, \tag{9}$$

  où chacun des coefficients est développé par la formule de Stirling (sauf bien sûr si $\alpha$ est entier).

Dans tous les exemples qui suivent, la fonction génératrice est connue explicitement. Cependant, il est important de noter qu'aucune de ces étapes ne nécessite cette connaissance explicite (voir exercice 8).

Exemple 15. Un dérangement de $n$ objets est une permutation où aucun des objets ne retrouve sa place initiale. Le nombre de dérangements sur un ensemble à $n$ éléments vaut

$$c_n = \frac{n!}{2!} - \frac{n!}{3!} + \frac{n!}{4!} + \cdots + (-1)^n.$$

La fonction génératrice $\sum_{n\geq 0} c_n z^n/n!$ vaut donc

$$\frac{e^{-z}}{1-z}.$$

La seule singularité de cette fonction est en 1, le développement au voisinage de 1 est évident :

```
series(exp(-z)/(1-z),z=1,2);
```

$$\frac{e^{-1}}{1-z} + e^{-1} + O(1-z)$$

dont on déduit

$$c_n = n!\big(e^{-1} + O(1/n^2)\big),$$

le $O$ étant ici très pessimiste. Pour $n = 10$ l'estimation asymptotique est déjà très bonne :

```
convert([seq((-1)^i*10!/i!,i=2..10)],`+`),exp(-1.)*10!;
```

$$1334961, \quad 1334960.916$$

Exemple 16. Si la commande `ztrans` était bien faite, son application à la récurrence

$$u_{n+1} = u_n + \sum_{i=0}^{n} u_i u_{n-1-i}, \qquad u_0 = 1$$

étudiée à la fin du chapitre VII donnerait l'équation suivante satisfaite par la fonction génératrice :

$$(U(z) - 1)/z = U(z) + U^2(z).$$

L'équation se résout pour donner la valeur de $U(z)$ :

```
solve((U(z)-1)/z=U(z)+U(z)^2,U(z));
```

$$\frac{1}{2z} - 1/2 - \frac{\sqrt{1-6z+z^2}}{2z}, \quad \frac{1}{2z} - 1/2 + \frac{\sqrt{1-6z+z^2}}{2z}$$

Seule la première de ces deux solutions est régulière à l'origine, c'est donc elle la fonction génératrice. Les singularités sont les zéros de la racine carrée :

```
fg:="[1]:
solve(1-6*z+z^2,z);
```

$$3 + 2\sqrt{2}, \quad 3 - 2\sqrt{2}$$

La singularité de plus petit module est la seconde racine. Il faut calculer le développement de la fonction génératrice en ce point. Idéalement il faudrait exécuter `series(subs(z="[2]*(1-t),fg),t,3)`. Mais `series` gère mal les constantes et il est nécessaire d'aider Maple. Nous décrivons une approche adaptée à la plupart des fonctions génératrices algébriques (c'est-à-dire racines d'un polynôme en $y$ et $z$). Sur l'équation définissant la fonction $U(z)$, les singularités sont facilement repérables : ce sont les points où le théorème des fonctions implicites cesse de s'appliquer. Au voisinage de ces points, on effectue un changement de variables en utilisant le fait que le développement attendu est en puissances demi-entières. Il ne reste plus qu'à appliquer une méthode de coefficients indéterminés comme en §2.3.

```
eq:=numer((U-1)/z-(U+U^2)):
assign(solve(subs(U=U0,z=z0,{eq,diff(eq,U)}),{z0,U0})):
evala(Normal(numer(subs(z=z0*(1-t^2),U=U0+V,eq)))):
s:=series(subs(V=convert([seq(a[i]*t^i,i=1..5)],`+`),"),t,7):
solve(coeff(s,t,2),a[1]);
```

$$-\frac{\sqrt{2}}{2}\sqrt{17-3\,\mathrm{RootOf}(1-6\_Z+\_Z^2)},\frac{\sqrt{2}}{2}\sqrt{17-3\,\mathrm{RootOf}(1-6\_Z+\_Z^2)}$$

Les éléments de la suite $u_n$ étant tous positifs, la fonction $U(z)$ est croissante jusqu'à sa singularité et donc, d'après les normalisations effectuées, c'est la racine négative qu'il faut considérer. Tous les autres coefficients s'en déduisent, d'où le développement cherché en utilisant (9) :

```
a[1]:="[1]:
for i from 2 to 5 do a[i]:=solve(coeff(s,t,i+1),a[i]) od:
r:=asympt(convert([seq(a[2*i+1]*GAMMA(n-i-1/2)/
    GAMMA(-i-1/2),i=0..2)],`+`)/n!,n,4):
simplify(subs(z0=3-2*sqrt(2),z0^(-n)*
    map(evala,map(Normal,combine(map(normal,r),exp)))));
```

$$(3-2\sqrt{2})^{(-n)}\left[\frac{\sqrt{4+3\sqrt{2}}}{2\sqrt{\pi}n^{3/2}}-\frac{3\left(25+18\sqrt{2}\right)}{32\sqrt{4+3\sqrt{2}}\sqrt{\pi}n^{5/2}}+\frac{5\left(964+687\sqrt{2}\right)}{2048\sqrt{4+3\sqrt{2}}\sqrt{\pi}n^{7/2}}+O\left(n^{-9/2}\right)\right]$$

### 3.8. Exercices.

1. Calculer le développement asymptotique lorsque $n\in\mathbb{N}$ tend vers l'infini de $\sin(2\pi en!)$. Vérifier numériquement le développement obtenu.
2. En utilisant le résultat de l'exercice 3 p. 204, calculer le comportement asymptotique de la valeur du terme maximum de la série du sinus intégral de Fresnel, lorsque $x$ tend vers l'infini. Calculer ensuite la valeur de $N$ telle que la valeur absolue du $N^{\mathrm{e}}$ terme de la série p. 189 vaille $\epsilon$, $\epsilon$ tendant vers 0.
3. Calculer le développement asymptotique de la fonction $y(x)$ définie par $y+\log y=x$, lorsque $x$ tend vers $+\infty$.
4. Nature de $\sum_{n=1}^{\infty}u_n$, où
$$u_n=\frac{1}{n^\alpha}\sum_{k=1}^{n}\log^2 k.$$
5. Calculer le développement asymptotique de
$$\sum_{n\geq 1}(1+1/n)^{n^2}\frac{x^n}{n!}$$
lorsque $x\to\infty$.
6. Soit $A_n$ le nombre d'entiers $i\in\{1,\dots,n\}$ tels que le reste de la division de $n$ par $i$ vaut au moins $i/2$. Donner un équivalent de $A_n$. [Indication : considérer une somme de Riemann.]

7. Calculer le développement asymptotique lorsque $x \to \infty$ de

$$\int_x^\infty \frac{\log t}{t^2} dt, \qquad \int_0^{\pi/2} e^{x\cos(t)}\, dt, \quad \int_0^1 (1-t^2)^x\, dt,$$
$$\int_0^1 [\log(1+t)]^x\, dt, \quad \int_0^\pi \sin t \cdot t^x\, dt, \quad \int_{-\infty}^{+\infty} e^{-xt^2}\sqrt{1+t^2}\, dt.$$

8. Déterminer le comportement asymptotique du $n^{\text{e}}$ coefficient de Taylor de la fonction $T(z)$ définie par

$$T(z) = 1 + zT^2(z) + z^2T^5(z).$$

9. On souhaite déterminer la limite quand $n \to \infty$ de la primitive

$$t \mapsto \int_0^t \frac{[(n+1)c_{n+1}e^x - nc_n]e^{1/(1+e^x)}}{e^{nx}(e^x+1)^2} dx,$$

où la suite $c_n$ est définie par $c_0 = 1$, $c_1 = -1$ et

$$nc_n + (2n-1)c_{n-1} + (n-2)c_{n-2} = 0, \qquad n > 1.$$

(1) Calculer et résoudre l'équation différentielle satisfaite par la fonction génératrice de la suite $\{c_n\}$ ;
(2) en déduire la fonction génératrice de l'intégrande puis de l'intégrale ;
(3) localiser la singularité dominante en fonction de $t$ et en déduire la limite.

CHAPTER IX

# Intégrales et primitives

Le calcul d'intégrales et de primitives est un domaine auquel le calcul formel a beaucoup à apporter. En effet, obtenir l'expression d'une primitive ou d'une intégrale définie est une opération en général difficile à réaliser à la main, autant en raison de la taille des calculs qu'en raison des connaissances mathématiques nécessaires. Les classes d'intégrales que les systèmes savent calculer sont bien délimitées et leur connaissance permet de guider les calculs. Nous décrivons ces classes dans ce chapitre. Nous commençons par les calculs de primitives, puis nous abordons les intégrales définies, et nous concluons le chapitre par le cas des intégrales dépendant d'un paramètre.

## 1. Primitives

Autant le calcul de dérivées est souvent réalisable à la main lorsque l'expression n'est pas très compliquée, autant celui de primitives est généralement difficile. Pour une fonction aussi simple que $1/(x^3+1)$ le calcul manuel d'une primitive est loin d'être immédiat. En revanche un système de calcul formel donne immédiatement une réponse :

```
int(1/(1+x^3),x);
```

$$\frac{1}{3}\ln(x+1)-\frac{1}{6}\ln(x^2-x+1)+\frac{1}{3}\sqrt{3}\arctan\left(\frac{1}{3}(2x-1)\sqrt{3}\right)$$

Même le recours à des tables de primitives n'est ni pratique ni toujours très sûr (certains citent le chiffre de plus de 10% d'erreurs dans ces tables !).

Les méthodes utilisées habituellement pour calculer à la main des primitives sont plutôt de nature heuristique : table des intégrales connues, changement de variables, intégration par parties, découpage de l'intégrande, reconnaissance de motifs... Ces méthodes permettent souvent de trouver une primitive assez rapidement, mais en aucun cas elles ne permettent de conclure en cas d'échec. Les techniques employées par les systèmes de calcul formel sont d'une autre nature. Les heuristiques du calcul manuel sont employées dans un premier temps, pour leur rapidité. Puis des *algorithmes* basés sur des théorèmes d'algèbre différentielle prennent le relais. Alors, pour des classes bien définies d'expressions, sur lesquelles nous allons revenir, la réponse négative du système est plus qu'un constat d'échec : il s'agit d'une *preuve* que la fonction n'a pas de primitive élémentaire. Le sens précis du terme élémentaire sera défini plus bas.

Nous allons passer en revue dans les paragraphes suivants les différentes classes de fonctions qu'un système de calcul formel sait intégrer.

**1.1. Fractions rationnelles.** Pour calculer l'intégrale d'une fraction rationnelle $p(x)/q(x)$, la méthode classique consiste à la décomposer en éléments simples, c'est-à-dire si le corps de base est $\mathbb{C}$, à la mettre sous la forme :

$$\frac{p(x)}{q(x)} = b(x) + \sum_{i=1}^{r} \sum_{j=1}^{\alpha_i} \frac{A_{i,j}}{(x-a_i)^j}$$

où $b(x)$ est un polynôme. L'intégration ne pose alors aucun problème. La seule difficulté, et elle est de taille, consiste à trouver les $a_i$, pôles de la fraction rationnelle, ce qui revient à factoriser $q(x)$ sur $\mathbb{C}$. Bien que les systèmes de calcul formel soient capables de factoriser les polynômes sur $\mathbb{Q}$, la factorisation sur $\mathbb{C}$ est une opération extrêmement coûteuse qui de plus n'est pas nécessaire à l'intégration.

L'intégration des fractions rationnelles pose deux problèmes aux systèmes de calcul formel. D'une part il faut éviter de faire trop grossir les expressions. Par exemple, Maple calcule la primitive de

$$\frac{5x^4+60x^3+255x^2+450x+275}{x^5+15x^4+85x^3+225x^2+274x+120}$$

sous la forme :

```
int((5*x^4+60*x^3+255*x^2+450*x+275)/
    (x^5+15*x^4+85*x^3+225*x^2+274*x+120),x);
```

$$\frac{25\ln(x+5)}{24} + \frac{5\ln(x+4)}{6} + \frac{5\ln(x+3)}{4} + \frac{5\ln(x+2)}{6} + \frac{25\ln(x+1)}{24}$$

Il est également possible d'exprimer cette primitive en fonction du logarithme d'un seul polynôme, mais ce polynôme est de degré 120 et ses coefficients sont de l'ordre de $10^{68}$.

Le second problème est illustré par la fraction rationnelle $1/(z^2-2)$. Sa primitive vaut :

```
int(1/(z^2-2),z);
```

$$-\frac{\sqrt{2}}{2}\operatorname{arctanh}\left(\frac{z\sqrt{2}}{2}\right)$$

Le nombre $\sqrt{2}$ apparaît dans le résultat et on peut montrer qu'il n'existe pas de primitive de cette fonction où ce nombre n'apparaît pas (à moins d'utiliser l'exponentielle). Dans le cas général, il n'est pas possible d'intégrer des fractions rationnelles sans faire appel à des nombres algébriques. Une des difficultés consiste alors à exprimer le résultat avec des nombres algébriques de degré le plus petit possible.

Exemple 1. La décomposition en éléments simples sur $\mathbb{Q}$ de $1/(x^6+1)$ ne donne pas les pôles de $f(x)$ car ils ne sont pas rationnels. Les nombres algébriques qui interviennent sont tous de degré 2 :

```
f:=1/(1+x^6);
```

$$f := \frac{1}{1+x^6}$$

```
int(f,x);
```

$$\frac{1}{3}\arctan(x) - \frac{1}{12}\sqrt{3}\ln(x^2-\sqrt{3}x+1) + \frac{1}{6}\arctan(2x-\sqrt{3}) + \frac{1}{12}\sqrt{3}\ln(x^2+\sqrt{3}x+1) + \frac{1}{6}\arctan(2x+\sqrt{3})$$

EXEMPLE 2. Le polynôme $x^5+x^2+2$ est irréductible sur $\mathbb{Q}$. La primitive de $(x^6+1)/(x^5+x^2+2)$ fait donc intervenir des nombres algébriques de degré 5. Les systèmes de calcul formel qui ne disposent pas de nombres algébriques sont par conséquent incapables de la calculer.

```
f:=(1+x^6)/(2+x^2+x^5);
```

$$f := \frac{1+x^6}{2+x^2+x^5}$$

```
int(f,x);
```

$$\frac{1}{2}x^2 + \sum_{\_R=\%1} \_R \ln\Big(x - \frac{55244064845502}{251979825667}\_R^4 + \frac{1183145201920}{251979825667}\_R^3 + \frac{25042172070863}{1007919302668}\_R^2 + \frac{3205221300291}{251979825667}\_R + \frac{440585345352}{251979825667}\Big)$$

$$\%1 := \text{RootOf}(50216\_Z^5 - 2133\_Z^3 - 3476\_Z^2 - 632\_Z - 104)$$

La sommation dans la solution est une somme sur toutes les racines du polynôme argument de `RootOf`. Pour vérifier le résultat il suffit de le dériver :

```
normal(diff(",x));
```

$$\frac{1+x^6}{2+x^2+x^5}$$

Ce résultat symbolique peut assister l'évaluation numérique de l'intégrale. Il suffit de calculer numériquement les racines du polynôme et la formule ci-dessus permet alors un calcul rapide et à précision arbitraire de l'intégrale.

EXEMPLE 3. L'exemple suivant illustre la capacité de l'algorithme à rechercher le résultat dans les extensions algébriques du plus petit degré possible.

```
f:=(7*x^13+10*x^8+4*x^7-7*x^6-4*x^3-4*x^2+3*x+3)/
   (x^14-2*x^8-2*x^7-2*x^4-4*x^3-x^2+2*x+1);
```

$$f := \frac{7x^{13}+10x^8+4x^7-7x^6-4x^3-4x^2+3x+3}{x^{14}-2x^8-2x^7-2x^4-4x^3-x^2+2x+1}$$

```
convert(f,parfrac,x);
```

$$\frac{7x^{13}+10x^8+4x^7-7x^6-4x^3-4x^2+3x+3}{x^{14}-2x^8-2x^7-2x^4-4x^3-x^2+2x+1}$$

```
int(f,x);
```

$$\frac{1}{2}\ln(x^7-\sqrt{2}x^2+(-\sqrt{2}-1)x-1)\sqrt{2}+\frac{1}{2}\ln(x^7-\sqrt{2}x^2+(-\sqrt{2}-1)x-1)$$
$$+\frac{1}{2}\ln(x^7+\sqrt{2}x^2+(\sqrt{2}-1)x-1)-\frac{1}{2}\ln(x^7+\sqrt{2}x^2+(\sqrt{2}-1)x-1)\sqrt{2}$$

Dans cet exemple, le dénominateur est irréductible de degré 14, mais les nombres algébriques nécessaires à l'expression de la solution sont de degré au plus 2, résultat difficile à obtenir sans un système de calcul formel.

En conclusion, toutes les fractions rationnelles à coefficients dans un corps $K$ admettent une primitive exprimée à l'aide d'une fraction rationnelle à coefficients dans $K$ et d'une somme de logarithmes de polynômes à coefficients dans une extension algébrique de $K$.

En pratique, lorsque $K=\mathbb{Q}$, Maple parvient toujours au résultat et il devrait toujours y parvenir lorsque $K$ est une extension algébrique de $\mathbb{Q}$.

Exemple 4. Voici un exemple où le corps de base est $\mathbb{Q}(\sqrt{2},\pi)$ :

```
f:=1/(sqrt(2)+Pi*x+x^5);
```

$$f := \frac{1}{\sqrt{2}+\pi x+x^5}$$

```
int(f,x);
```

$$\sum_{\_R=\%1} \_R\ln\left((4\pi\_R+1)x+5\sqrt{2}\_R\right)$$
$$\%1 := \text{RootOf}\left((256\pi^5+12500)\_Z^5-160\_Z^3\pi^3-80\_Z^2\pi^2-15\_Z\pi-1\right)$$

Pour calculer l'intégrale d'une fraction rationnelle où interviennent des nombres flottants, Maple commence par convertir ces nombres en rationnels, puis effectue le calcul et le résultat obtenu est transformé à nouveau en nombres flottants. Lorsque `Digits` est élevé, cette conversion est parfois coûteuse.

Exemple 5.

```
f:=(1.2*x+1)/(2.3*x^2+x-0.1);
```

$$f := \frac{1.2x+1}{2.3x^2+x-.1}$$

```
int(f,x);
```

$$.2608695652\ln(23.x^2+10.x-1.)$$
$$-1.066842889\,\text{arctanh}(3.319764049x+.7216878367)$$

**1.2. Fonctions élémentaires.** La classe des fractions rationnelles est incluse dans la classe des fonctions élémentaires. Pour toutes les fonctions de cette dernière classe, le calcul formel dispose d'un algorithme de recherche de primitive. L'algorithme mène soit à une primitive soit à la preuve qu'une primitive élémentaire n'existe pas.

Les fonctions élémentaires sont les fonctions bâties à partir des fractions rationnelles à coefficients dans un corps $K$ (en pratique $K=\mathbb{Q}$) par application répétée de l'exponentielle, du logarithme, des opérations $+$, $\times$, $-$, $\div$ et de l'opération de clôture algébrique.

Par exemple l'ensemble des fonctions élémentaires sur $\mathbb{Q}(x)$ contient la fonction $\sqrt[3]{x^3-1}$, solution de l'équation $y^3-x^3+1=0$ et la fonction $\ln(x^5+x+1)$. Il contient aussi les fonctions trigonométriques car, par exemple,

$$\sin x = \frac{e^{ix}-e^{-ix}}{2i},$$

$i$ étant obtenu comme solution de $y^2+1=0$. De même les fonctions trigonométriques inverses sont élémentaires. Par exemple,

$$\arctan x = \frac{i}{2}\ln\left(\frac{i+x}{i-x}\right).$$

Pour une fonction élémentaire, le principe de Liouville (env. 1875) donne la forme de sa primitive si cette dernière est elle aussi élémentaire et le théorème de structure de Risch (1969) fournit un algorithme qui permet de décider si une fonction élémentaire a une primitive élémentaire ou non.

En pratique, l'implantation de Maple est complète pour les extensions purement transcendantes, c'est-à-dire celles où aucune fonction algébrique n'intervient. Dans ce cas, l'échec du système prouve l'inexistence d'une primitive élémentaire. Dans le cas des extensions algébriques, l'implantation est encore incomplète, certaines primitives sont trouvées par le système, mais il échoue parfois alors qu'une primitive élémentaire existe.

Exemple 6. Voici un exemple de fonction purement transcendante admettant une primitive élémentaire. Le système la trouve sans peine :

```
f:=exp(x^3+1)*(3*x^3-1)/x^2;
```

$$f := \frac{e^{(x^3+1)}(3x^3-1)}{x^2}$$

```
int(f,x);
```

$$\frac{e^{(x^3+1)}}{x}$$

Exemple 7. Voici une autre fonction purement transcendante :

```
int(1/(ln(x^3+1)),x);
```

$$\int \frac{1}{\ln(x^3+1)}\,dx$$

Ce résultat non évalué *prouve* que $1/\ln(x^3+1)$ n'a pas de primitive élémentaire.

Exemple 8. Le cas de l'intégration d'une fonction élémentaire algébrique est plus difficile. Nous avons déjà mentionné que Maple ne peut intégrer toutes les fonctions algébriques. En outre, le programme nécessite parfois l'emploi de la notation `RootOf`. La commande `alias` simplifie les expressions.

```
f:=(x^2+1)^(1/3)/x:
int(f,x);
```

$$\int \frac{(x^2+1)^{1/3}}{x}dx$$

```
alias(alpha=RootOf(z^3-x^2-1,z)): f:=convert(f,RootOf);
```

$$f := \frac{\alpha}{x}$$

```
int(f,x);
```

$$\frac{3}{2}\alpha - \frac{3}{2}\ln\Big((-48\%1 - 55 - 9\%1^2 + 9\%1^2x^2 + 27x^2\%1 - 22x^2 - 45\alpha\%1 - 54\alpha - 45\%1\alpha^2 - 54\alpha^2)/x^2\Big)\,\%1 - \frac{1}{2}\ln\Big((-48\%1 - 55 - 9\%1^2 + 9\%1^2x^2 + 27x^2\%1 - 22x^2 - 45\alpha\%1 - 54\alpha - 45\%1\alpha^2 - 54\alpha^2)/x^2\Big) + \frac{3}{2}\%1\ln\Big(-(-8 + 18\%1 - 9\%1^2 - 6x^2 + 3x^2\%1 + 9\%1^2x^2 + 21\alpha\%1 - 7\alpha + 21\%1\alpha^2 - 7\alpha^2/x^2)$$

$$\%1 := \mathrm{RootOf}(9\_Z^2 + 3\_Z + 1)$$

Cet exemple illustre une difficulté d'utilisation des intégrateurs actuels : l'évaluation numérique (par exemple en $x = 1$) produit une partie imaginaire non nulle, ce qui n'est généralement pas souhaité.

Exemple 9. Voici une fonction algébrique admettant une primitive élémentaire que Maple ne parvient pas à trouver :

$$\int \frac{dx}{(2x-1)^3\sqrt[3]{x(x-1)^2}}.$$

Pour parvenir à la calculer, il suffit d'effectuer le changement de variable

$$t = \sqrt[3]{\frac{x-1}{x}}.$$

Nous verrons en §1.4 comment s'y prendre.

À l'heure actuelle, le seul système de calcul formel qui implante complètement l'algorithme de Risch est Axiom.

**1.3. Autres fonctions.** Un certain nombre d'intégrales simples ne s'expriment pas à l'aide des fonctions élémentaires. Par leur fréquente apparition et leur importance dans certaines applications, quelques-unes d'entre elles ont reçu un nom et font partie des *fonctions spéciales*. Les plus importantes de celles que connaît Maple sont la fonction d'erreur, l'exponentielle intégrale, le sinus intégral et le cosinus intégral, soit respectivement

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,dt, \qquad \mathrm{Ei}(n,x) = \int_1^\infty \frac{e^{-xt}}{t^n}\,dt, \qquad \mathrm{Si}(x) = \int_0^x \frac{\sin t}{t}\,dt$$

$$\mathrm{Ci}(x) = \gamma + \ln(ix) - \frac{i\pi}{2} + \int_0^x \frac{\cos t - 1}{t}\,dt.$$

Deux autres "fonctions" importantes pour l'intégration sont reconnues par Maple : la fonction `Heaviside(t)` qui est nulle pour $t < 0$ et vaut 1 pour $t \geq 0$, et sa dérivée la fonction `Dirac(t)`.

Toutes ces fonctions sont connues du système, en ce sens qu'il est possible de les évaluer numériquement, de les dériver, d'en calculer le développement en série,... Il est également possible d'en intégrer certaines. Il ne s'agit plus là d'une classe d'expressions pour laquelle l'intégration est garantie. Ceci signifie que lorsque Maple ne trouve pas de primitive d'une expression comportant des fonctions spéciales, cela n'équivaut pas à une preuve de l'inexistence d'une primitive dans cette classe. Cependant quelques heuristiques permettent au système d'intégrer certaines de ces fonctions ou de les utiliser pour exprimer des primitives de fonctions élémentaires.

EXEMPLE 10. Nous profitons ici de la différence entre la commande d'intégration `int` et sa forme inerte `Int`.

```
f:=exp(x^2+1)/x:
Int(f,x) = int(f,x);
```

$$\int \frac{e^{(x^2+1)}}{x} = -\frac{1}{2}\,\mathrm{Ei}(1,-x^2)e$$

```
f:=sin(x^2+1)/x:
Int(f,x) = int(f,x);
```

$$\int \frac{\sin(x^2+1)}{x} = \frac{1}{2}\cos(1)\,\mathrm{Si}(x^2) + \frac{1}{2}\sin(1)\,\mathrm{Ci}(x^2)$$

```
int(Heaviside(x)*x^2,x);
```

$$\frac{1}{3}x^3\,\mathrm{Heaviside}(x)$$

```
int(Dirac(x)*Dirac(x-t),x);
```

$$\mathrm{Heaviside}(x)\,\mathrm{Dirac}(-t)$$

EXEMPLE 11. D'autres fonctions spéciales, comme les fonctions de BESSEL, sont traitées par des heuristiques du même type :

```
int(x^4*BesselJ(2,x)*BesselJ(1,x),x);
```

$$\frac{1}{8}x^5(\mathrm{BesselJ}(2,x)\,\mathrm{BesselJ}(1,x) + \mathrm{BesselJ}(3,x)\,\mathrm{BesselJ}(2,x))$$

**1.4. Commandes de récriture.** Lorsque la fonction à intégrer (ou sa primitive) n'est pas élémentaire, et que les heuristiques ont échoué, il se peut que l'intégrale, ou une partie, puisse être ramenée par récriture dans une classe de fonctions que le système sait intégrer. Les deux opérations principales de récriture sont le changement de variable et l'intégration par parties. Ces techniques s'appliquent également aux fonctions algébriques qui, quoiqu'élémentaires, celles-ci ne sont pas encore bien intégrées par Maple.

1.4.1. *Changement de variable*

Sous de bonnes conditions de différentiabilité le changement de variable $x = \varphi(t)$ permet de transformer

$$F(x) = \int f(x)\,dx \qquad \text{en} \qquad F[\varphi(t)] = \int f[\varphi(t)]\varphi'(t)\,dt.$$

La commande Maple correspondante se nomme `changevar` et se trouve dans le *package* `student`.

Exemple 12. Maple ne sait pas calculer la primitive suivante :

```
e:=1/(1+((x+1)/x)^(1/3)): int(e,x);
```

$$\int \frac{1}{1+\left(\dfrac{x+1}{x}\right)^{1/3}}\,dx$$

Or les expressions de la forme

$$\int R\left(x,\sqrt[n]{\frac{ax+b}{cx+d}}\right)\,dx$$

où $R(u,v)$ est une fraction rationnelle en $u$ et $v$ ($a$, $b$, $c$, $d$ étant des constantes et $n$ un entier) se ramènent à des primitives de fractions rationnelles par le changement de variable

$$t=\sqrt[n]{\frac{ax+b}{cx+d}}.$$

```
student[changevar]((x+1)/x=t^3,Int(e,x),t);
```

$$\int -3\frac{t^2}{(1+t)(1-t^3)^2}\,dt$$

```
value(");
```

$$-\frac{3}{4}\ln(1+t)+\frac{1}{6}\frac{1}{t-1}+\frac{1}{12}\ln(t-1)+\frac{1}{3}\ln(t^2+t+1)+\frac{1}{3}\frac{t-1}{t^2+t+1}$$

Les fonctions `Int` et `value` ont été présentées au §2.3.5 du chapitre I. Pour trouver la primitive cherchée, il suffit de remplacer dans la valeur obtenue $t$ par

$$\sqrt[3]{\frac{x+1}{x}}.$$

1.4.2. *Intégration par parties*

La fonction arcsin est élémentaire, mais n'est pas purement transcendante à cause de la racine carrée :

$$\arcsin x=-i\ln(\sqrt{1-x^2}+ix).$$

Maple obtient facilement la primitive de $\arcsin x$ :

```
int(arcsin(x),x);
```

$$x\arcsin(x)+\sqrt{1-x^2}$$

En revanche il ne trouve pas celle de $\arcsin^2 x$ :

```
intarc:=int(arcsin(x)^2,x);
```

$$intarc:=\int \arcsin(x)^2\,dx$$

Comme la dérivée de $\arcsin x$ est $1/\sqrt{1-x^2}$, l'intégration par parties vient facilement à bout de cette intégrale et plus généralement de celle de $\arcsin^n x$.

La formule de l'intégration par parties, en supposant qu'il n'y ait pas de problème de différentiabilité, est :

$$\int u(x)v'(x)dx=u(x)v(x)-\int u'(x)v(x)dx$$

Il faut donc donner à la commande `student[intparts]` d'intégration par parties l'intégrale à calculer et la partie $u(x)$ à dériver (qui n'apparaît pas obligatoirement, Maple divisant l'intégrande par $u(x)$ pour trouver la partie à intégrer). L'intégration de $\arcsin^2 x$ se mène alors facilement en réalisant deux intégrations par parties successives.

```
student[intparts](intarc,arcsin(x)^2);
```

$$\arcsin(x)^2 x - \int 2\frac{\arcsin(x)x}{\sqrt{1-x^2}}dx$$

```
student[intparts](",arcsin(x));
```

$$\arcsin(x)^2 x + 2\arcsin(x)\sqrt{1-x^2} + \int -2\,dx$$

```
value(");
```

$$\arcsin(x)^2 x + 2\arcsin(x)\sqrt{1-x^2} - 2x$$

Ce processus peut être répété autant de fois que nécessaire pour calculer la primitive de puissances plus élevées de $\arcsin x$. Par exemple pour calculer la primitive de $\arcsin^4 x$, on intégrera par parties en prenant $u(x) = \arcsin^4 x$, puis une nouvelle fois avec $u(x) = \arcsin^3 x$ et ainsi de suite, la dernière intégration par parties étant effectuée avec $u(x) = \arcsin x$.

Le calcul de l'intégrale dans le cas de $n$ entier quelconque sera repris au §3.2.

## 2. Intégrales définies

En matière d'intégration numérique, la connaissance d'une primitive permet de réduire les temps de calculs et d'éviter les problèmes d'instabilité numérique.

Par exemple si l'on doit calculer une intégrale du type

$$\int_{a(x)}^{b(x)} f(x,y)dy$$

pour un grand nombre de valeurs de $x$ et que la primitive a une expression analytique, il est bien plus efficace de passer du temps à calculer cette expression une fois pour toutes puis à l'évaluer ensuite numériquement plutôt que de faire chaque fois l'intégration numérique.

Cette situation se produit en particulier lorsque, dans une intégrale multiple, on peut intégrer formellement une partie de l'intégrale, comme dans l'exemple suivant. Le temps de l'intégration formelle est très court et la connaissance de la primitive permet de gagner un facteur 20 dans l'évaluation numérique.

Exemple 13.

```
i:=Int(Int(sqrt(2-x^2-y^2),y=0..x),x=0..1);
```

$$i := \int_0^1 \int_0^x \sqrt{2-x^2-y^2}\,dy\,dx$$

```
st:=time(): evalf(i),time()-st;
```

$$.5685164493, \quad 113.400$$

```
i:=int(int(sqrt(2-x^2-y^2),y=0..x),x=0..1);
```

$$i := \int_0^1 \frac{1}{2}\sqrt{2-2x^2}x + \frac{1}{2}I\ln(Ix+\sqrt{2-2x^2})x^2 - I\ln(Ix+\sqrt{2-2x^2}) + I\ln(\sqrt{2-x^2}) - \frac{1}{2}I\ln(\sqrt{2-x^2})x^2 dx$$

```
st:=time(): evalf(i),time()-st;
```

$$.5685164493 + .550826106810^{-14}I, \quad 5.967$$

Dans cette section, nous passons en revue les différentes classes d'intégrales définies que Maple sait calculer : soit à partir d'une primitive, soit par recherche dans une table, soit par transformée de Laplace ou de Fourier. Nous traitons ensuite les intégrales multiples et l'intégration numérique.

**2.1. Utilisation d'une primitive.** Pour calculer

$$\int_a^b f(x)\,dx,$$

la méthode classique consiste à calculer une primitive $F(x)$ de $f(x)$, puis à effectuer $F(b) - F(a)$, ou bien dans le cas des intégrales impropres

$$\lim_{x\to b^-} F(x) - \lim_{x\to a^+} F(x).$$

Il est à noter que ce dernier cas ne pose en général pas de problème particulier. De plus, Maple sait calculer un certain nombre d'intégrales impropres autre que celles dont il sait calculer la primitive (voir le §2.2).

Cette méthode n'est pas valide sans quelques hypothèses sur $f$. La vérification de ces hypothèses pose parfois problème aux systèmes de calcul formel. Les deux principales difficultés sont la détection de singularités dans l'intervalle d'intégration et le choix de la bonne branche des fonctions multiformes.

2.1.1. *Singularités dans l'intervalle d'intégration*

L'exemple le plus simple est celui de

$$\int_{-1}^1 \frac{1}{x^2}\,dx.$$

```
f:=1/x^2;F:=int(f,x):
int(f,x=-1..1),subs(x=1,F)-subs(x=-1,F);
```

$$\infty, \quad -2$$

La formule $F(b)-F(a)$ ne s'applique pas à cause du pôle en 0, mais Maple trouve le bon résultat car il commence par essayer de repérer les discontinuités en utilisant la fonction `discont`. La puissance de cette commande est limitée par celle de `solve`. Il faut donc faire attention lorsque les singularités de la fonction n'ont pas de forme analytique ou lorsque la fonction à intégrer dépend d'un paramètre.

EXEMPLE 14. Le dénominateur $z^2\exp(z) - 2z - 1$ s'annule deux fois sur l'intervalle $[-1, 2]$, mais Maple ne s'en rend pas compte, et fournit un résultat alors que la fonction n'est pas intégrable sur cet intervalle.

```
f:=(2*z*exp(z)+z^2*exp(z)-2)/(z^2*exp(z)-2*z-1):
int(f,z=-1..2);
```

$$\ln(4e^2 - 5) - \ln(e^{-1} + 1)$$

EXEMPLE 15. Le calcul suivant n'est valable que pour $a \notin [-1, 1]$.

```
int(1/(x-a)^2,x=-1..1);
```

$$\frac{1}{-1+a} - \frac{1}{1+a}$$

EXEMPLE 16. Il est à noter que même lorsque la fonction $f$ est continue, la primitive $F$ trouvée peut ne pas l'être, ce qui complique le calcul d'intégrales définies :

```
f:=1/(3+2*cos(x)): F:=int(f,x);
```

$$F := \frac{2}{5}\sqrt{5}\arctan\left(\frac{1}{5}\tan(\frac{1}{2}x)\sqrt{5}\right)$$

```
int(f,x=0..2*Pi), eval(subs(x=2*Pi,F)-subs(x=0,F));
```

$$\frac{2}{5}\sqrt{5}\pi, \quad 0$$

Ici, Maple a détecté la discontinuité de $F$ en $x = \pi$, et le résultat donné par `int` est correct.

### 2.1.2. *Fonctions multiformes*

Les techniques utilisées pour calculer des primitives prêtent peu attention au domaine de définition de la fonction à intégrer. La formule renvoyée par le système est correcte, en ce sens que sa dérivée formelle est bien la fonction intégrée. Cependant, cette primitive peut ne pas être définie sur tout l'intervalle d'intégration. Dans les exemples que nous avons vus ci-dessus, la primitive cessait d'être définie à cause de la présence d'une singularité dans l'intervalle. Une autre cause fréquente d'incorrection est un changement de détermination d'une fonction multiforme.

EXEMPLE 17. Le résultat suivant est faux :

```
f:=t->sin(t)*cos(t)*ln(tan(t)):
Int(f(t),t=0..Pi/2)=evalc(int(f(t),t=0..Pi/2));
```

$$\int_0^{\pi/2} \sin(t)\cos(t)\ln(\tan(t))\,dt = -\frac{1}{2}I\pi$$

La bonne valeur est 0, car $f(\pi/4 + u)$ est impaire :

```
simplify(f(Pi/4+u)+f(Pi/4-u));
```

$$0$$

Lorsque l'intégrande lui-même contient une fonction multiforme, Maple choisit l'une des formes *a priori* :

```
f:=sqrt(1-cos(x)^2): F:=int(f,x);
```

$$F := \cos(x)$$

Ici la primitive $\cos(x)$, qui correspond à l'intégrande $-\sin(x)$, n'est correcte que pour $(2k-1)\pi \leq x \leq 2k\pi$. Sur les autres intervalles, cette primitive donne des résultats aberrants :

```
int(f,x=0..Pi);
```

$$-2$$

2.1.3. *Fonctions complexes*

Un autre cas de fonctions multiformes peut apparaître lors de l'intégration de fonctions complexes. En effet, les fonctions obtenues lors de l'intégration peuvent par exemple être définies à $k\pi$ près, et le système de calcul formel peut très bien prendre des déterminations non compatibles lors des calculs.

C'est le cas pour l'exemple suivant :

Exemple 18. Calculons

$$\int_{-\infty}^{+\infty} \frac{dx}{1+(x+2i)^2}.$$

Maple donne le résultat suivant :

```
f:=1/(1+(x+2*I)^2):
int(f,x=-infinity..infinity);
```

$$\pi$$

Ce résultat est faux ! Pour le vérifier, nous calculons séparément l'intégrale de la partie réelle et de la partie imaginaire :

```
evalc(f);
```

$$\frac{-3+x^2}{(-3+x^2)^2+16x^2} - 4\frac{Ix}{(-3+x^2)^2+16x^2}$$

```
map(int,",x=-infinity..infinity);
```

$$0$$

L'explication de l'erreur se trouve en calculant la primitive et sa limite pour $x$ tendant vers $-\infty$ et $+\infty$ :

```
int(f,x);
```

$$\arctan(x+2I)$$

```
limit(",x=infinity), limit(",x=-infinity);
```

$$\frac{1}{2}\pi, -\frac{1}{2}\pi$$

```
evalc(map(int,evalc(f),x));
```

$$\begin{aligned} &-\frac{1}{2}\arctan(x) + \frac{1}{2}\arctan(\frac{1}{3}x) - \frac{1}{4}(\frac{1}{2} - \frac{1}{2}\,\mathrm{signum}(x^2+9))\pi \\ &\quad + \frac{1}{4}(\frac{1}{2} - \frac{1}{2}\,\mathrm{signum}(x^2+1))\pi \\ &\quad + I(\frac{1}{4}\ln(|x^2+9|) - \frac{1}{4}\ln(|x^2+1|)) \end{aligned}$$

```
limit(",x=infinity), limit(",x=-infinity);
```

$$0, 0$$

Cet arctan complexe se récrit en terme de logarithme :

$$\frac{1}{2} i \ln(\frac{3-ix}{-1+ix})$$

formule à laquelle on peut arriver en simplifiant le résultat de

```
convert(arctan(x+2*I),ln);
```

Il est alors clair par composition des limites que la limite est la même en $\pm\infty$ en prenant les mêmes déterminations principales pour le logarithme complexe. Maple semble ignorer dans son calcul que l'argument de arctan est un nombre complexe.

**2.2. Classes d'intégrales définies.** Outre les fonctions dont il sait déterminer une primitive, Maple sait calculer plusieurs classes d'intégrales définies sans passer par le calcul de la primitive. C'est en particulier ainsi que le système calcule de nombreuses intégrales impropres.

Les classes d'intégrales définies reconnues évoluent de version en version. La technique est simplement basée sur une recherche dans des tables de formules, où il ne reste plus qu'à déterminer les valeurs des paramètres. En outre, une cinquantaine d'intégrales définies élémentaires sont stockées dans la table de *remember* de la fonction `int/def`.

Voici les classes d'intégrales reconnues par la version actuelle (ce qui veut dire que dans ces cas, des conditions supplémentaires sur les paramètres sont testées, qui peuvent ou non mener à une valeur) :

$$\int_{-\infty}^{\infty} P(x) \exp(ax^2 + bx + c)\, dx, \tag{1}$$

où $P(x)$ est un polynôme en $x$ ;

$$\int_{x}^{\infty \text{ ou } 1} \frac{\exp(u_1 t^{s_1} + u_2 t^{s_2}) t^v \ln^m(bt^{d_1}) \cos(ct^r)}{(\alpha + \beta f^d)^p}\, dt, \tag{2}$$

où $f$ est soit $t$ soit $\exp(wt)$, et le cosinus peut être remplacé par un sinus ;

$$\int_{0}^{\infty} \exp(ut^s) t^v \ln^m(bt^{d_1}) \operatorname{erf}(ft^v + g)\, dt, \tag{3}$$

où $f$ est soit $t$ soit $\exp(wt)$ ;

$$\int_{0}^{\infty} \exp(ut^s) t^w J_{k_1}^{p_1}(ct^d) Y_{k_2}^{p_2}(ct^d)\, dt, \tag{4}$$

où $J_k$ et $Y_k$ sont les fonctions $J$ et $Y$ de Bessel d'indice $k$ ;

$$\int_{\alpha}^{\beta} \cos(a \sin t) \cos(bt)\, dt, \tag{5}$$

où les différents cosinus et sinus peuvent être intervertis.

Une autre classe est celle des intégrales elliptiques, dont l'intégrande est une fraction rationnelle $R(x, s(x))$, où $s(x)$ est soit la racine carrée d'un

polynôme de degré inférieur à 5, soit la racine cubique ou quatrième d'un polynôme de degré 2.

EXEMPLE 19. Dans cet exemple, la formule (2) s'applique avec $v = -1/3$, $u_1 = u_2 = c = 0$, $\alpha = d = m = b = d_1 = 1$, $\beta = -1$, $p = -1/2$ :

```
f:=(x^(-1/3))*ln(x)*((1-x)^(1/2)): int(f,x);
```

$$\int \frac{\ln(x)\sqrt{1-x}}{x^{1/3}}\,dx$$

```
normal(int(f,x=0..1));
```

$$\frac{3}{98}\,\frac{\left(-42\,\gamma + 7\,\pi\,\sqrt{3} - 63\,\ln(3) - 288 - 42\,\Psi(\frac{1}{6})\right)\Gamma(\frac{2}{3})\,\Gamma(\frac{5}{6})}{\sqrt{\pi}}$$

où $\Psi$ est la dérivée logarithmique de la fonction $\Gamma$.

EXEMPLE 20. Ici encore la formule (2) s'applique :

```
f:=exp(-x^2+1)/x;
```

$$f := \frac{e^{(1-x^2)}}{x}$$

```
int(f,x=1..infinity);
```

$$\frac{1}{2}e\mathrm{Ei}(1,1)$$

Un tel résultat permet en particulier une évaluation numérique avec une précision arbitraire

```
evalf(",30);
```

.298173681161597037170539249684

EXEMPLE 21. Cette intégrale elliptique est donnée en terme de la fonction $F$ de LEGENDRE :

```
f:=1/sqrt((1-x^2)*(1-4*x^2));
```

$$f := \frac{1}{\sqrt{(1-x^2)(1-4x^2)}}$$

```
int(f,x=1..3);
```

$$\frac{1}{2}\mathrm{LegendreF}(1,\frac{1}{2}) - \frac{1}{2}\mathrm{LegendreF}(\frac{1}{3},\frac{1}{2})$$

```
evalf(");
```

.6721475221

**2.3. Méthode des résidus.** La méthode des résidus peut servir à calculer certaines intégrales impropres.

Supposons en effet que l'on veuille calculer l'intégrale

$$\int_0^\infty \frac{\cos x}{x^2+a^2}dx, \qquad a > 0.$$

Maple donne un résultat complexe insatisfaisant :

```
assume(a>0):int(cos(x)/(x^2+a^2),x=0..infinity);
```

$$-\frac{1}{2}\,\frac{I\pi(-I\sinh(a\tilde{})+\cosh(a\tilde{})+I\cosh(a\tilde{}))}{a\tilde{}} + \frac{1}{2}\,\frac{\cosh(a\tilde{})\pi}{a\tilde{}}$$

```
Im(");
```

$$-\frac{1}{2}\frac{\cosh(a\tilde{})\pi}{a\tilde{}}$$

Si l'on remarque que cette intégrale est égale à

$$\frac{1}{2}\int_{-\infty}^{\infty}\frac{e^{ix}}{x^2+a^2}dx,$$

la fonction $\sin x$ étant impaire, Maple donne 0 comme résultat, ce qui est faux (voir le §2.1.2).

```
f:=exp(I*x)/(x^2+a^2)/2;
```

$$f := \frac{1}{2}\frac{e^{(Ix)}}{x^2+a\tilde{}^2}$$

```
int(f,x=-infinity..infinity)/2;
```

$$0$$

On peut utiliser la méthode des résidus pour calculer cette intégrale. On considère la fonction complexe correspondante

$$f(z)=\frac{1}{2}\frac{e^{iz}}{z^2+a^2}.$$

Les seuls pôles de la fonction étant les points $z=ai$ et $z=-ai$ de l'axe imaginaire, on peut intégrer $f(z)$ selon un lacet $\gamma$ formé du segment réel $[-R,R]$ avec $R>a$ et du demi cercle $\gamma_1$ de centre $O$ et de rayon $R$ situé dans le demi-plan des parties imaginaires positifs. On a alors :

$$\int_\gamma f(z)dz=\int_{-R}^{R}f(x)dx+\int_{\gamma_1}f(z)dz=2\pi i\ \mathrm{Res}_{ai}f$$

$$\text{avec}\quad \int_{\gamma_1}f(z)dz=\int_0^\pi f(Re^{it})d(Re^{it})$$

La valeur cherchée est alors

$$\lim_{R\to+\infty}\int_{-R}^{R}f(x)dx.$$

Tout ces calculs se font aisément en Maple. La fonction `residue` (qu'il faut charger) permet de calculer le résidu.

```
f1:=subs(x=R*exp(I*t),f)*diff(R*exp(I*t),t);
```

$$f1 := \frac{1}{2}\frac{Ie^{(IRe^{(It)})}Re^{(It)}}{R^2(e^{(It)})^2+a\tilde{}^2}$$

```
int(f1,t=0..Pi);
```

$$\frac{1}{4}\frac{I(-e^{a\tilde{}}\,\mathrm{Ei}(1,a\tilde{}+IR)+e^{(-a\tilde{})}\,\mathrm{Ei}(1,IR-a\tilde{}))}{a\tilde{}}$$
$$-\frac{1}{4}\frac{I(-e^{a\tilde{}}\,\mathrm{Ei}(1,-IR+a\tilde{})+e^{(-a\tilde{})}\,\mathrm{Ei}(1,-IR-a\tilde{}))}{a\tilde{}}$$

```
limit(",R=infinity);
```

$$0$$

```
readlib(residue)(f,x=a*I)*2*Pi*I;
```

$$\frac{1}{2}\frac{e^{(-a\tilde{})}\pi}{a\tilde{}}$$

Et on obtient le résultat cherché qui est égal à $\dfrac{\pi}{2a}e^{-a}$.

**2.4. Transformées intégrales.** Les transformées intégrales sont un cas particulier d'intégrales impropres à un paramètre que le système "calcule" en recherchant dans une table. Par une aberration d'organisation du logiciel, la fonction `int` ne sait pas reconnaître ces intégrales définies ; c'est donc à l'utilisateur de les identifier.

Les transformées les plus connues sont la transformée de LAPLACE et la transformée de FOURIER qui sont respectivement définies par

$$L(s) = \int_0^\infty f(t)e^{-st}\,dt \qquad \text{et} \qquad F(s) = \int_{-\infty}^\infty f(t)e^{-ist}\,dt.$$

Ces transformées sont très utiles dans des domaines comme l'automatique ou le traitement du signal. Leur calcul en Maple, lorsqu'il est possible, est immédiat avec les fonctions `laplace` et `fourier`.

EXEMPLE 22.

```
laplace((Heaviside(t)*sin(t))/t+sin(3*t)/t,t,s);
```

$$\pi - \arctan(s) - \arctan(s/3)$$

Pour obtenir ce résultat avec `int`, il faut préciser que $s$ est positif ou nul et intégrer chaque terme séparément.

La transformée de LAPLACE permet entre autres de résoudre des équations différentielles ou intégrales.

EXEMPLE 23. Pour résoudre l'équation intégrale

$$\sin t = 1 + \int_0^t e^x f(t-x)\,dx, \qquad t > 0$$

en $f(t)$, utilisons la transformée de LAPLACE et sa transformée inverse.

```
eq:=sin(t)=1+int(exp(x)*f(t-x),x=0..t);
```

$$eq := \sin(t) = 1 + \int_0^t e^x f(t-x)dx$$

```
laplace(eq,t,s);
```

$$\frac{1}{1+s^2} = \frac{1}{s} + \frac{\text{laplace}(f(t),t,s)}{s-1}$$

```
solve(",laplace(f(t),t,s));
```

$$\left(\frac{1}{1+s^2} - \frac{1}{s}\right)(s-1)$$

```
sol:=invlaplace(",s,t);
```

$$sol := -\,\text{Dirac}(t) + 1 + \cos(t) - \sin(t)$$

Vérifions la cohérence du résultat :

```
subs(f(t-x)=subs(t=t-x,sol),eq);
```

$$\sin(t) = 1 + \int_0^t e^x \left(-\,\text{Dirac}(t-x) + 1 + \cos(t-x) - \sin(t-x)\right)\,dx$$

```
assume(t>0):simplify(",Dirac);
```
$$\sin(t\tilde{}) = \sin(t\tilde{})$$

Il est à noter que Maple ne sait pas calculer

$$\int_0^t \delta(t-x)f(x)dx$$

pour une fonction $f(x)$ quelconque tant que l'on n'a pas spécifié que $t$ est positif.

Une utilisation marginale de la transformée de LAPLACE est le calcul de certaines intégrales contenant des fonctions spéciales. Comme dans l'exemple ci-dessus, il suffit de calculer la transformée de LAPLACE de l'intégrale, puis la transformée inverse.

EXEMPLE 24. L'intégrale à calculer est :

$$\int_0^t \left(J_0(x) - xJ_1(x)\right) dx,$$

où $J_i$ est la fonction $J$ de BESSEL d'indice $i$.

```
f:=BesselJ(0,x)-x*BesselJ(1,x);
```
$$f := \mathrm{BesselJ}(0,x) - x\,\mathrm{BesselJ}(1,x)$$

```
int(f,x=0..t);
```
$$\int_0^t \left(\mathrm{BesselJ}(0,x) - x\,\mathrm{BesselJ}(1,x)\right) dx$$

```
normal(laplace(",t,s));
```
$$\frac{s}{(1+s^2)^{3/2}}$$

```
invlaplace(",s,t);
```
$$t\,\mathrm{BesselJ}(0,t)$$

La transformée de FOURIER est une méthode simple et efficace pour résoudre les équations différentielles linéaires rencontrées en électronique ou en mécanique.

EXEMPLE 25. Le mouvement d'un oscillateur forcé (par exemple un dipôle RLC soumis à une tension sinusoïdale) est régi par l'équation

```
eq := diff(x(t),t,t)+2*lambda*diff(x(t),t)+omega[0]*x(t)
      = f[0]*cos(omega*t);
```
$$eq := \left(\frac{d^2}{dt^2}x(t)\right) + 2\lambda\left(\frac{d}{dt}x(t)\right) + \omega_0 x(t) = f_0\cos(\omega\, t)$$

et la résolution via la transformée de FOURIER se mène comme suit :

```
readlib(fourier)(eq,t,s);
```
$$-s^2\,\mathrm{fourier}(x(t),t,s) + 2I\lambda s\,\mathrm{fourier}(x(t),t,s) + \omega_0\,\mathrm{fourier}(x(t),t,s) = f_0\pi(\mathrm{Dirac}(s-\omega) + \mathrm{Dirac}(s+\omega))$$

```
solve(",fourier(x(t),t,s));
```
$$-\frac{-f_0\pi\,\mathrm{Dirac}(s-\omega) - f_0\pi\,\mathrm{Dirac}(s+\omega)}{-s^2 + 2I\lambda s + \omega_0}$$

```
normal(evalc(invfourier(",s,t)));
```

$$\frac{f_0\left(-\cos(\omega t)\omega^2+\cos(\omega t)\omega_0+2\lambda\omega\sin(\omega t)\right)}{\omega^4-2\omega^2\omega_0+{\omega_0}^2+4\lambda^2\omega^2}$$

L'expression trouvée est une solution particulière de l'équation différentielle `eq` ; on pourra comparer à la solution générale obtenue par `dsolve(eq,x(t))`.

Maple connaît aussi la transformée de Mellin définie par

$$M(s)=\int_0^\infty f(t)t^{s-1}dt,$$

qui a de nombreux usages en théorie analytique des nombres et en analyse asymptotique.

```
mellin(t/(t^2+1),t,s);
```

$$\frac{1}{2}\frac{\pi}{\sin\left(\pi(\frac{1}{2}s+\frac{1}{2})\right)}$$

**2.5. Intégrales multiples.** Les intégrales multiples, doubles ou triples, interviennent en physique ou en mécanique pour calculer des aires, des volumes, des masses, des moments d'inertie,...

Les systèmes de calcul formel ne disposent généralement pas d'intégrateur multiple. Il faut utiliser plusieurs fois l'intégration simple. L'ordre des intégrations peut jouer un rôle important et l'utilisateur doit le définir.

Exemple 26. Pour calculer l'intégrale double

$$\iint (x^2+y^2)\,dx\,dy$$

sur le domaine défini par $(0<x<h,\ y^2-2px<0)$, ce qui revient à calculer

$$\int_0^h \int_{-\sqrt{2px}}^{\sqrt{2px}} (x^2+y^2)\,dy\,dx,$$

il faut exécuter en Maple:

```
int(int(x^2+y^2,y=-sqrt(2*p*x)..sqrt(2*p*x)),x=0..h);
```

$$\frac{4}{105}\frac{(ph)^{5/2}\sqrt{2}(15h+14p)}{p^2}$$

Exemple 27. Le calcul du volume engendré par un arc $AB$ tournant autour de l'axe $Oz$ se réduit à l'intégrale curviligne $\pi\int_{AB} x^2\,dz$. Pour un arc $AB$ paramétré par $(x=f(t),\ z=g(t))$, cela donne

$$\pi\int_{t_1}^{t_2} f^2(t)g'(t)\,dt.$$

Par exemple, le volume du domaine illimité engendré par la cissoïde $z^2x=(2a-x)^3$ lorsqu'elle tourne autour de l'axe $Oz$ (fig. 1) s'obtient en utilisant la représentation paramétrique suivante de la cissoïde :

$$x=f(t)=a(1-\cos t),\quad z=g(t)=a\frac{(1+\cos t)^2}{\sin t}\qquad t\in\,]0,2\pi[\,.$$

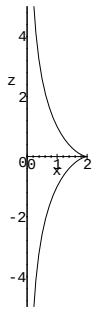



FIGURE 1 La cissoïde $z^2x = (2a - x)^3$ pour $a = 1$.

La courbe est symétrique par rapport à l'axe $Ox$ et la valeur du volume en découle :

```
f:=a*(1-cos(t)): g:=a*(1+cos(t))^2/sin(t):
Pi*int(f^2*diff(g,t),t=2*Pi..0);
```

$$2\pi^2 a^3$$

Il ne faut cependant pas être trop optimiste concernant l'usage du calcul formel pour les intégrales multiples. Dans la plupart des cas, le système ne parvient pas à calculer une forme explicite. Cependant, comme nous l'avons déjà mentionné, l'obtention d'une formule pour un ou deux niveaux d'intégration présente l'intérêt de réduire considérablement la durée d'une évaluation numérique.

**2.6. Intégration numérique.** Tous les systèmes de calcul formel permettent de faire de l'intégration numérique. En Maple, il suffit d'utiliser la fonction `evalf` appliquée à la forme inerte de l'intégrale `Int(f,x=a..b)`, ou à ce que retourne Maple lorsqu'il ne sait pas calculer l'intégrale définie.

EXEMPLE 28.

```
int(exp(sin(x)),x=-Pi..Pi);
```

$$\int_{-\pi}^{\pi} e^{\sin(x)} dx$$

```
evalf(");
```

$$7.954926521$$

Il vaut cependant mieux utiliser directement la forme inerte, car cela évite la perte de temps liée à la recherche d'une primitive explicite.

Pour ces intégrations numériques, Maple emploie par défaut la quadrature de CLENSHAW-CURTIS là où l'intégrande est régulier, et traite les singularités par développement limité. Il est parfois utile de choisir une des deux autres méthodes disponibles (une procédure de quadrature exponentielle et la méthode de NEWTON-COTES) afin d'avoir des temps de calcul moins longs.

Exemple 29. Un fleuve de débit 1 se divise en deux, le débit du premier sous-fleuve étant déterminé par une loi uniforme, puis chaque sous-fleuve se redivise en deux suivant la même règle. Le calcul de l'espérance du débit du plus gros des quatre sous-sous-fleuves mène à une intégrale multiple faisant intervenir la fonction max.

Si nous essayons de faire le calcul avec la méthode d'intégration par défaut (quadrature de Clenshaw-Curtis), Maple s'arrête au bout de 95 minutes, en disant qu'il n'arrive pas à traiter les singularités. En revanche, avec la règle de Newton-Cotes, spécifiée par l'argument `_NCrule`, le résultat est obtenu en moins de deux minutes :

```
f := max(x*y,x*(1-y),(1-x)*z,(1-x)*(1-z));
```

$$f := \max(x(1-y), (1-x)z, xy, (1-x)(1-z))$$

```
st:=time():
evalf(Int(Int(Int(f,x=0..1,10,_NCrule),y=0..1,10,_NCrule),
                    z=0..1,10,_NCrule)),time()-st;
```

$$.5659985785, \quad 117.716$$

Ce résultat est correct, à une unité près sur la dernière décimale.

L'évaluation numérique des intégrales multiples est assez lente en Maple. Il faut savoir en effet que du fait de la méthode employée, le temps de calcul est multiplié par 25 environ à chaque dimension supplémentaire. Dans le cas d'une fonction régulière, la méthode de Simpson peut donner rapidement une première approximation :

```
st:=time():
student[simpson](f,x=0..1,8):
student[simpson](",y=0..1,8):
student[simpson](",z=0..1,8):
evalf("), time()-st;
```

$$.5661168983, \quad 8.917$$

Le dernier argument de `simpson` est le nombre d'intervalles utilisés ; la fonction a été évaluée ici $9^3 = 729$ fois.

2.6.1. *Fonctions oscillantes*

Dans le cas de fonctions changeant de signe un nombre très grand — voire infini — de fois sur l'intervalle d'intégration, les méthodes classiques d'intégration numérique échouent. Il est parfois possible de mener à bien le calcul en assistant le système. Illustrons la méthode sur une intégrale produite par Maple lors du développement limité d'une intégrale paramétrée (ch. VIII, exemple 8 p. 203) :

```
f:=t->ln(1+t)/(1+t):
Int(f(t)*sin(t),t=0..infinity);
```

$$\int_0^\infty \frac{\ln(1+t)\sin(t)}{1+t}\,dt$$

```
evalf(");
Error, (in evalf/int) singularity in or near
interval of integration
```

Le principe consiste à découper l'intervalle d'intégration en régions où l'intégrande se comporte de façon régulière. Dans notre exemple, sur chaque intervalle $[2k\pi, 2(k+1)\pi]$, l'intégrale de $f(t)\sin(t)$ est la même que celle de $(f(2k\pi+u)-f(2k\pi))\sin(u)$ sur $[0,2\pi]$ à cause de la périodicité du sinus. On commence donc par calculer un développement asymptotique de $f(2k\pi+u)-f(2k\pi)$ lorsque $k$ tend vers l'infini :

```
h:=combine(map(normal,asympt(f(2*k*Pi+u)-f(2*k*Pi),k,4)),ln);
```

$$h := -\frac{1}{4}\frac{u\,(-1+\ln(2k\pi))}{\pi^2k^2} + \frac{1}{16}\frac{u\left(-6-3u+u\ln(k^2)+u\ln(4\pi^2)+\ln(16\pi^4k^4)\right)}{\pi^3k^3} + O\left(\frac{1}{k^4}\right)$$

puis on intègre ce développement multiplié par $\sin(u)$ entre 0 et $2\pi$ :

```
i:=combine(map(normal,asympt(int(h*sin(u),u=0..2*Pi),k)),ln);
```

$$i := \frac{1}{2}\frac{-1+\ln(2k\pi)}{\pi k^2} - \frac{1}{8}\frac{4\ln(\pi)\pi+\ln(16)\pi-6\pi+\pi\ln(k^4)-6+\ln(16\pi^4k^4)}{\pi^2k^3} + O\left(\frac{1}{k^4}\right)$$

Ceci nous indique que l'intégrale $I_k$ de $f(t)\sin(t)$ entre $2k\pi$ et $2(k+1)\pi$ se comporte comme $\ln k/k^2$. La somme des contributions est donc convergente, et on peut même obtenir un développement asymptotique de $I_k + I_{k+1} + I_{k+2} + \ldots$ grâce à la formule d'Euler-Maclaurin :

```
combine(map(normal,asympt(-eulermac(i,k),k)),ln);
```

$$\frac{\ln(\sqrt{2}\sqrt{k\pi})}{k\pi} - \frac{1}{16}\frac{\ln(16)\pi+\pi\ln(\frac{1}{16\pi^4})+4\ln(\pi)\pi-4+\ln(16\pi^4k^4)}{\pi^2k^2} + O\left(\frac{1}{k^3}\right)$$

En combinant ce développement avec une intégration numérique sur l'intervalle $[0, 2k\pi]$, on obtient une approximation de l'intégrale :

```
evalf(subs(k=100,Int(f(t)*sin(t),t=0..2*k*Pi)+"));
```

$$.3494748858 + O(\frac{1}{1000000})$$

Cette méthode s'applique aux intégrales de la forme $\int_a^\infty f(t)g(t)dt$, où $g$ est une fonction périodique de moyenne nulle, et $f$ est une fonction à variation lente. Suivant le comportement de $f$ à l'infini, l'intégrale diverge ou converge. Dans ce dernier cas, il faut trouver un compromis entre l'ordre des développements asymptotiques et la valeur de $k$ de façon à obtenir une approximation avec la précision voulue.

## 3. Intégrales paramétrées

Du point de vue du calcul formel, la présence de paramètres correspond à une extension du corps de base. Les transformées intégrales sont d'ailleurs un cas particulier d'intégrales paramétrées, où le paramètre $s$ appartient à un domaine bien défini (réel, réel positif ou complexe). Tant que les paramètres $a_1, a_2, \dots$ n'interviennent que dans des fractions rationnelles, le corps de base usuel $\mathbb{Q}$ est remplacé par $\mathbb{Q}(a_1, a_2, \dots)$ dans lequel la nullité reste décidable, en supposant $a_1, a_2, \dots$ algébriquement indépendants (voir chap. I, §2.3.1). Généralement les programmes sont plus lents mais le calcul de primitive fonctionne encore. Le calcul d'intégrale définie est bien plus délicat puisqu'il nécessite la détection de singularités sur le domaine d'intégration. L'usage de la commande `assume` devient alors crucial pour mener à bien les simplifications nécessaires.

### 3.1. Cas général.

Exemple 30. Dans cet exemple, le paramètre $t$ est positif. Il faut le signifier assez tôt au système, car sinon il va exprimer l'intégrale en termes de `signum` et parfois de `limit`, fonctions difficiles à manier et à simplifier dans des expressions compliquées.

```
f:=1/(t^2*cos(x)^2+sin(x)^2);
```

$$f := \frac{1}{t^2\cos(x)^2 + \sin(x)^2}$$

```
r:=Int(f,x=0..Pi): value(r);
```

$$2\frac{\mathrm{csgn}(\sqrt{t^2}\sqrt{4 + \mathrm{conjugate}(2t^2 - 2\sqrt{t^4})})\pi}{\sqrt{t^2}\sqrt{2t^2 - 2\sqrt{t^4} + 4}}$$

```
assume(t>0): value(r);
```

$$\frac{\pi}{t^{\sim}}$$

Il est également possible de manipuler des objets plus formels. Voici une intégrale dépendant d'un paramètre :

```
g:=int(f(x,t),x=a(t)..b(t));
```

$$g := \int_{a(t)}^{b(t)} f(x,t)\,dx$$

Et voici le calcul de sa dérivée :

```
diff(g,t);
```

$$\int_{a(t)}^{b(t)} \frac{\partial}{\partial t} f(x,t)dx + f(b(t),t)\frac{\partial}{\partial t}b(t) - f(a(t),t)\frac{\partial}{\partial t}a(t)$$

Il faut noter que ce calcul est purement formel et qu'une utilisation pratique de ce résultat nécessite la vérification de certaines hypothèses de continuité et de dérivabilité.

**3.2. Suites d'intégrales.** Les suites d'intégrales peuvent être vues comme un cas particulier d'intégrales dépendant d'un paramètre où celui-ci ne prend que des valeurs entières positives.

Pour les calculer, une technique souvent employée consiste à chercher des formules de récurrence entre les intégrales de la suite. Une autre technique plus simple lorsqu'elle s'applique consiste à employer des fonctions génératrices. Nous détaillons cette seconde technique sur un exemple de calcul de série de FOURIER au chapitre VIII, page 203.

EXEMPLE 31. Nous avons déjà vu que la fonction arcsin n'est pas purement transcendante, ce qui cause des difficultés à l'intégrateur de Maple. La primitive de $\arcsin^n x$ pour $n$ entier supérieur ou égal à 2 ne s'obtient pas directement, mais est fournie par intégrations par parties répétées. La procédure suivante réalise ce calcul :

```
intarcsin:=proc(n:posint)
local e,i;
   e:=Int(arcsin(x)^n,x);
   for i from n by -1 to 1 do e:=student[intparts](e,arcsin(x)^i) od;
   value(e)
end:
intarcsin(5);
```

$$\arcsin(x)^5 x + 5\arcsin(x)^4\sqrt{1-x^2} - 20\arcsin(x)^3 x - 60\arcsin(x)^2\sqrt{1-x^2} + 120x\arcsin(x) + 120\sqrt{1-x^2}$$

La raison pour laquelle cette technique fonctionne si bien est que ces intégrales vérifient une récurrence linéaire d'ordre 2 :

```
student[intparts](Int(arcsin(x)^n,x),arcsin(x)^n);
```

$$\arcsin(x)^n x - \int \frac{\arcsin(x)^n n x}{\sqrt{1-x^2}\arcsin(x)}\,dx$$

```
student[intparts](simplify(",power),arcsin(x)^(n-1));
```

$$\arcsin(x)^n x + \arcsin(x)^{(n-1)}\sqrt{1-x^2}n + \int -\frac{\arcsin(x)^{(n-1)}(n-1)n}{\arcsin(x)}\,dx$$

Autrement dit, en notant $J_n$ la primitive de $\arcsin^n x$, ces intégrales vérifient la récurrence :

$$J_n = x\arcsin^n x + n\sqrt{1-x^2}\arcsin^{n-1} x - n(n-1)J_{n-2}.$$

Il suffit alors de savoir calculer $J_0 = x$ et $J_1 = x\arcsin(x) + \sqrt{1-x^2}$ pour trouver la forme générale de ces intégrales.

EXEMPLE 32. La même méthode s'applique à l'intégrale

$$I_n = \int \frac{dx}{(x^2+1)^n}.$$

En outre dans cet exemple, la commande `rsolve` permet de résoudre la récurrence.

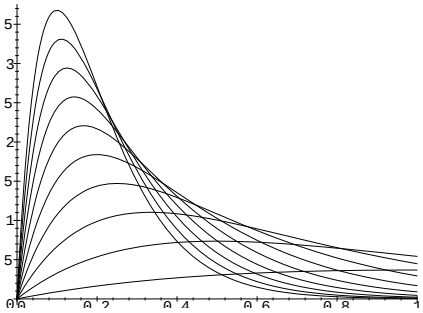


FIGURE 2 La suite $f_n(x) = n^2 x e^{-nx}$.

```
int(1/(1+x^2)^n,x);
```

$$\int \frac{1}{(1+x^2)^n} dx$$

```
student[intparts](",1/(1+x^2)^n);
```

$$\frac{x}{(1+x^2)^n} - \int \frac{-2nx^2}{(1+x^2)^n(1+x^2)}\, dx$$

La récurrence est donc une récurrence linéaire d'ordre 1 :

$$I_n = \frac{x}{(x^2+1)^n} + 2nI_n - 2nI_{n+1}$$

```
rsolve({u(n)=x/(x^2+1)^n+2*n*u(n)-2*n*u(n+1),
   u(1)=arctan(x)},u(n));
```

$$\frac{\Gamma(n-1/2)}{2\sqrt{\pi}\Gamma(n)} \left( x\sqrt{\pi} \sum_{\_n2=1}^{n-1} \frac{\left(x^2+1\right)^{-\_n2} \Gamma(\_n2)}{\Gamma(\_n2+1/2)} + 2\arctan(x) \right)$$

Dans tous ces calculs, il faut faire attention aux justifications. Ainsi, la limite d'une suite d'intégrales n'est pas toujours l'intégrale de la limite, comme l'illustre l'exemple suivant.

EXEMPLE 33. La suite de fonctions $f_n(x) = n^2 x e^{-nx}$ converge vers la fonction nulle. Le tracé des courbes correspondantes pour diverses valeurs de $n$ montre que cette convergence n'est pas uniforme (fig. 2). Ce tracé est obtenu par

```
f:=n^2*x*exp(-n*x):
plot({seq(subs(n=k,f),k=1..10)},x=0..1);
```

Voici la limite des intégrales

```
limit(int(f,x=0..1),n=infinity);
```

$$1$$

et voici l'intégrale de la limite :

```
assume(x>0):int(limit(f,n=infinity),x=0..1);
```

$$0$$

### 3.3. Exercices.

1. Calculer l'intégrale $\displaystyle\int_{-\infty}^{+\infty} \frac{dx}{a^2+(x+ib)^2}$ où $a$ et $b$ sont des réels.
2. Calculer la primitive de l'exemple 9 à l'aide du changement de variable indiqué, puis la dériver et retrouver la forme de l'intégrande. [Indication : on pourra utiliser les commandes `factor`, `normal(...,expanded)` et `simplify(...,power,symbolic)`.]
3. Montrer que l'intégrale double de l'exemple 13 vaut $(5-2\sqrt{2})\pi/12$.
4. Refaire les mêmes opérations que pour la suite de fonctions $f_n(x) = n^2 x e^{-nx}$ du §3.2 (tracé de courbes, calcul des intégrales et des limites) lorsque :
$$f_n(x) = \begin{cases} 2n^2 x & \text{pour} \quad 0 \le x \le 1/(2n) \\ 2n(1-nx) & \text{pour} \quad 1/(2n) \le x \le 1/n \\ 0 & \text{pour} \quad 1/n \le x \le 1 \end{cases}.$$
5. Trouver une formule générale ou une récurrence pour les intégrales
$$\int \arccos^n x\, dx, \qquad \int_0^\infty \frac{\sin^n x}{x^n}\, dx, \qquad \int \ln^n x\, dx, \qquad n \in \mathbb{N}^\star.$$
6. L'intégrale double
$$\int_0^\infty \int_x^\infty \frac{\sin t}{t}\, dt\, dx$$
a une valeur simple. Aider un système de calcul formel à la calculer.
7. L'intégrale
$$\int_0^{\pi/4} \ln(1+\tan x)\, dx$$
admet une valeur simple ; la calculer.
8. Calculer
$$\int_{-1}^{1} \frac{dx}{\sqrt{1+x}+\sqrt{1-x}+2}$$
par un changement de variable approprié.
9. La primitive de $\sqrt{e^{-2x}+e^{-x}+1}$ est élémentaire. La calculer.
10. L'intégrale triple de l'exercice 29 vaut $\frac{17}{6}\ln(2) - \frac{3}{2}\ln(3) + \frac{1}{4}$. Le montrer à l'aide d'un système de calcul formel.

CHAPTER X

# Calcul différentiel

Le calcul différentiel est un domaine très vaste pour lequel le calcul formel peut rendre un grand nombre de services. En particulier, lors de la résolution des équations différentielles ordinaires un système de calcul formel peut venir à bout des problèmes de singularités mieux qu'un système numérique par la recherche de solutions exactes ou sous la forme de développements en série. C'est ce que nous montrerons en §1. En §2, nous indiquerons sur deux exemples comment un système de calcul formel peut traiter très agréablement des problèmes liés aux propriétés différentielles des courbes.

## 1. Équations différentielles ordinaires

Les équations différentielles ordinaires sont omniprésentes dans les sciences pures et appliquées. De nombreuses techniques ont été développées pour en rechercher des solutions approchées par le calcul numérique, mais ces méthodes se dégradent généralement loin des conditions initiales ou en présence de singularités. Bien que ne couvrant qu'un champ plus restreint, des méthodes formelles existent qui ne souffrent pas de ce défaut. Dans cette section nous présentons les outils numériques et formels que propose Maple à travers la commande `dsolve` et le *package* `DEtools` pour l'étude des équations différentielles.

Il existe aussi des techniques de recherche de solutions analytiques approchées d'équations différentielles ; les systèmes ne fournissent pas d'outil spécialisé, mais les méthodes s'apparentent à la recherche de développements de solutions, dont nous parlons. Il en va de même pour les équations aux dérivées partielles, qui sortent encore du champ d'application du calcul formel.

**1.1. Solutions exactes.** La recherche de solutions exactes, c'est-à-dire exprimées en termes de fonctions élémentaires (cf. chap. IX §1.2), consiste essentiellement à déterminer si l'équation appartient à l'une des classes d'équations que l'on sait au moins partiellement résoudre ou simplifier, ou si elle peut s'y ramener par un changement de variables. Les classes d'équations reconnues varient d'un système à l'autre et d'une version à l'autre. Cependant, les équations suivantes sont généralement reconnues ($y$ est la fonction inconnue, $x$ la variable de dérivation) :

- équations linéaires :

$$a_0(x)y^{(p)}(x) + a_1(x)y^{(p-1)}(x) + \cdots + a_p(x)y(x) = b(x);$$

- équations homogènes :

$$P(x, y(x))y'(x) + Q(x, y(x)) = 0,$$

avec $P$ et $Q$ homogènes en $x, y$ de même degré ;
- équations autonomes : $x$ n'apparaît pas dans l'équation ;
- équations de Riccati :

$$y'(x) = a(x)y^2(x) + b(x)y(x) + c(x);$$

- équations de Bernoulli :

$$y'(x) + P(x)y(x) = Q(x)y^n(x);$$

- équations de Clairaut :

$$f(xy'(x) - y(x)) = g(y'(x));$$

- équations d'Euler :

$$a_0x^ny^{(n)}(x) + a_1x^{n-1}y^{(n-1)}(x) + \cdots + a_ny(x) = 0.$$

Les systèmes de calcul formel possèdent des procédures spécialisées qui leur permettent de simplifier ces équations, parfois jusqu'à la solution. La plupart des exercices classiques sur les équations différentielles ordinaires sont ainsi résolus par les systèmes. Cela ne dispense cependant pas l'utilisateur de connaître ces équations, car lorsqu'un système n'arrive pas à résoudre, on ne bénéficie pas de ses efforts intermédiaires pour simplifier l'équation. Il faut alors faire les changements de variables adéquats à la main pour aider le système dans les dernières étapes, ou passer à une étude numérique. Nous allons dans cette section montrer sur des exemples comment arriver à une solution en interagissant avec Maple, en présentant au passage la syntaxe des différentes commandes d'aide à la résolution.

Exemple 1. L'équation suivante est *séparable* : on peut la récrire de façon que la fonction inconnue et la variable apparaissent chacune dans un seul membre de l'équation :

$$Ax(y')^2 - y^2 + 1 = 0, \qquad A > 0, \quad y(0) = 1. \tag{1}$$

La syntaxe pour résoudre cette équation en Maple est la suivante :

```
dsolve({A*x*diff(y(x),x)^2-y(x)^2+1=0,y(0)=1},y(x));
```

Comme bien trop souvent, Maple échoue sur cet exemple. En fait seule la condition initiale pose problème (dans ce cas précis, un message d'erreur dû à une faute de programmation est renvoyé, mais en général `dsolve` retourne la valeur `NULL` en cas d'échec, et Maple n'affiche rien). Nous recommençons donc sans la condition initiale :

```
dsolve(A*x*diff(y(x),x)^2-y(x)^2+1=0,y(x));
```

$$2\frac{\sqrt{Axy(x)^2-Ax}\sqrt{y(x)^2-1}}{Ay(x)^2-A}$$
$$+\int_0^{y(x)}\frac{y_1^4}{(y_1^2-1)^{5/2}}-\frac{2y_1^2}{(y_1^2-1)^{5/2}}+\frac{1}{(y_1^2-1)^{5/2}}\,dy_1=\_C_1,$$
$$-2\frac{\sqrt{Axy(x)^2-Ax}\sqrt{y(x)^2-1}}{Ay(x)^2-A}$$
$$+\int_0^{y(x)}\frac{y_1^4}{(y_1^2-1)^{5/2}}-\frac{2y_1^2}{(y_1^2-1)^{5/2}}+\frac{1}{(y_1^2-1)^{5/2}}\,dy_1=\_C_1$$

Les intégrales des membres gauches n'ont pas été calculées. Pour en déclencher le calcul, il suffit de demander l'évaluation de l'expression précédente (cf. chap. II). Pour simplifier, nous nous contentons de la première équation :

```
"[1];
```

$$2\frac{\sqrt{Axy(x)^2-Ax}\sqrt{y(x)^2-1}}{Ay(x)^2-A}+\ln(y(x)+\sqrt{y(x)^2-1})-\frac{i\pi}{2}=\_C_1$$

Cette équation se résout en $y(x)$. Nous allons détailler comment on aide Maple à obtenir la solution. Nous commençons par utiliser la condition initiale pour déterminer $\_C_1$ :

```
solve(subs(x=0,y(0)=1,"),_C1);
```

$$-\frac{i\pi}{2}$$

On isole ensuite le logarithme et on élève au carré :

```
op(1,op(1,""))^2=op(2,op(1,""))^2;
```

$$4\frac{(Axy(x)^2-Ax)(y(x)^2-1)}{(Ay(x)^2-A)^2}=\ln(y(x)+\sqrt{y(x)^2-1})^2$$

On peut alors normaliser le terme de gauche :

```
map(normal,");
```

$$4\frac{x}{A}=\ln\left(y(x)+\sqrt{y(x)^2-1}\right)^2$$

Il y a alors deux racines possibles :

```
epsilon*op(1,")^(1/2)=op(1,op(2,"));
```

$$\epsilon\sqrt{4}\sqrt{\frac{x}{A}}=\ln(y(x)+\sqrt{y(x)^2-1})$$

où $\epsilon$ vaut 1 ou $-1$. Il ne reste plus alors qu'à prendre l'exponentielle et à résoudre, puis à simplifier un peu :

```
solve(subs(y(x)=y,map(exp,")),y):
convert(simplify(expand("),exp),trig);
```

$$\cosh\left(2\epsilon\sqrt{\frac{x}{A}}\right)$$

Comme le cosinus hyperbolique est pair, on peut se passer de $\epsilon$. Lorsque $x$ est négatif, cette solution devient un cosinus :

```
assume(A>0,x<0);
simplify(subs(epsilon=1,"));
```

$$\cos\left(2\frac{\sqrt{-A\tilde{}x\tilde{}}}{A\tilde{}}\right)$$

La seconde solution de l'équation différentielle mène au même résultat.

EXEMPLE 2. Voici une équation *autonome* — la variable par rapport à laquelle on dérive n'intervient pas — qui nous permet de présenter la syntaxe des conditions initiales pour les dérivées de la fonction :

$$(y'')^2 = (1+y'^2)^3, \qquad y(0) = y'(0) = 1.$$

```
deq:=diff(y(x),x$2)^2=(1+diff(y(x),x)^2)^3:
dsolve({deq,y(0)=1,D(y)(0)=1},y(x));
```

$$y(x) = -i\sqrt{x^2-\sqrt{2}x-1/2}-\frac{\sqrt{2}}{2}+1, \quad y(x) = i\sqrt{x^2+\sqrt{2}x-1/2}+\frac{\sqrt{2}}{2}+1$$

Pour spécifier des conditions initiales d'ordre plus élevé, on utilise l'opérateur de composition itérée `@@`, par exemple `(D@@2)(y)(0)=1` pour $y''(0) = 1$.

EXEMPLE 3. Maple n'arrive pas à résoudre l'équation

$$y'' = yy'/x$$

```
eq:=diff(y(x),x,x)=y(x)*diff(y(x),x)/x:
dsolve(eq,y(x));
```

Cependant, comme cette équation est invariante par le changement de variable $x \mapsto ax$, on a intérêt à changer $x$ en $e^t$, ce qui la transforme en une équation autonome. Pour faire le changement de variable, il faut aider le système ainsi :

```
eval(subs(y(x)=Y(ln(x)),eq)):
subs(ln(x)=t,x=exp(t),"):
map(numer,");
```

$$D^{(2)}(Y)(t) - \mathrm{D}(Y)(t) = Y(t)\mathrm{D}(Y)(t)$$

La résolution fonctionne sur cette équation :

```
dsolve(",Y(t));
```

$$t = 2\frac{\arctan\left(\frac{Y(t)+1}{\sqrt{2\_C_1-1}}\right)}{\sqrt{2\_C_1-1}} - \_C_2$$

Il ne reste plus qu'à effectuer le changement de variable inverse :

```
subs(t=ln(x),solve(",Y(t)));
```

$$\left(\tan(1/2\ln(x)\sqrt{2\_C_1-1}+1/2\_C_2\sqrt{2\_C_1-1})-\frac{1}{\sqrt{2\_C_1-1}}\right)\sqrt{2\_C_1-1}$$

donc les solutions de $y'' = yy'/x$ sont de la forme $y = a\tan(a\ln(x)/2+b)-1$.

EXEMPLE 4. Les fonctions *liouvilliennes* sont celles que l'on peut obtenir en composant l'exponentielle, l'intégration, l'extraction de racines de polynômes, et les opérations de corps, à partir des fractions rationnelles en une variable à coefficients dans $\mathbb{Q}$. Maple dispose d'algorithmes pour trouver ces solutions dans le cas des équations différentielles linéaires à coefficients polynomiaux d'ordre 1 et 2.

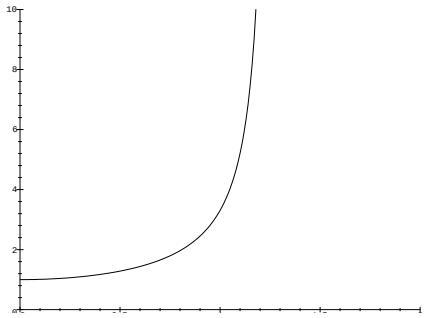


FIGURE 1

L'équation suivante rentre dans cette classe :

$$(x^3/2 - x^2)y''(x) + (2x^2 - 3x + 1)y'(x) + (x-1)y(x) = 0.$$

```
dsolve((x^3/2-x^2)*diff(y(x),x$2)
      +(2*x^2-3*x+1)*diff(y(x),x)+(x-1)*y(x),y(x));
```

$$y(x) = \frac{\_C_1 e^{-1/x}}{\sqrt{x}\sqrt{x-2}} + \frac{\_C_2 e^{-1/x} \int \frac{e^{1/x}\,dx}{x^{3/2}\sqrt{x-2}}}{\sqrt{x}\sqrt{x-2}}.$$

Comme pour la recherche de primitives, les algorithmes existant dans la littérature permettent de prouver qu'une équation n'admet pas de solution liouvillienne. Cependant, leur implantation n'est pas complète en Maple, et l'échec du programme ne constitue donc pas une preuve d'inexistence.

**1.2. Développements en série et asymptotiques.** Bien qu'il ne soit pas possible en général de résoudre exactement une équation différentielle, un certain nombre d'informations exactes peuvent être obtenues directement en partant de l'équation. C'est le cas en particulier pour les développements en série de TAYLOR et certains calculs asymptotiques. Les questions relatives à ces développements sont abordées au chapitre VIII ; nous ne donnons ici qu'un exemple supplémentaire.

EXEMPLE 5. On considère l'équation :

$$y' = \sin(x) + xy^2(x), \qquad y(0) = 1,$$

ce qui se traduit en Maple par :

```
deq:=diff(y(x),x)=sin(x)+x*y(x)^2:
```

La courbe correspondante est représentée en figure 1. Il y apparaît un comportement singulier vers $x \simeq 1,2$. Notre objectif est de trouver le développement de cette fonction au voisinage de cette singularité.

On ne sait pas calculer de solution explicite de cette équation, pas plus qu'une expression formelle de la singularité, qui dépend de la condition initiale $y(0) = 1$. Nous poursuivons donc le calcul en notant cette singularité par le symbole $\rho$. Nous adoptons un procédé heuristique qui consiste à essayer de déterminer si le comportement asymptotique de la solution n'est pas dicté

par un petit nombre de termes de l'équation. Dans notre exemple, $y(x)$ tend vers l'infini à sa singularité (cela se voit sur la figure 1 et il est possible de le prouver rigoureusement), et donc le terme en $\sin(x)$ devient négligeable, ce qui nous amène à considérer une équation plus simple :

```
dsolve(diff(y(x),x)=x*y(x)^2,y(x));
```

$$\frac{1}{y(x)} = -\frac{x^2}{2} + \_C_1$$

Pour satisfaire la condition $y(x) \to +\infty$ quand $x \to \rho$, il faut $\_C_1 = \rho^2/2$, ce qui donne $y(x) \sim K/(\rho - x)$ au voisinage de $\rho$. Ceci suggère d'effectuer dans l'équation de départ le changement $y(x) = 1/u(x)$ et de chercher un développement de $u(x)$ en série au voisinage de $x = \rho$. Pour cela, en Maple il faut d'abord ramener le point d'étude à l'origine. On obtient finalement le développement cherché par :

```
subs(x=rho-t,eval(subs(y(x)=1/U(rho-x),deq))):
series(subs(dsolve({",U(0)=0},U(t),series),1/U(t)),t,4):
subs(t=rho-x,");
```

$$-\frac{1}{\rho(x-\rho)} + \frac{1}{2\rho^2} - \frac{\left(-\frac{\sin(\rho)\rho}{3} + \frac{1}{4\rho^2}\right)(x-\rho)}{\rho} + O\big((x-\rho)^2\big)$$

En d'autres termes, après un premier changement de variable, l'équation différentielle est résolue en série à l'origine, puis le développement en série de l'inverse de la solution est calculé et il n'y a plus qu'à remettre la variable de départ dans ce développement.

**1.3. Méthodes numériques.** Il est bien rare qu'une équation différentielle se résolve explicitement, mais de nombreuses méthodes numériques permettent d'en étudier les solutions. Les systèmes de calcul formel, orientés vers les calculs exacts, ne sont pas les meilleurs outils pour aborder ces problèmes. Cependant, il peut se produire qu'une équation différentielle apparaisse au cours d'un calcul plus formel que numérique, et que l'on veuille poursuivre une étude interactive. Les systèmes tirent aussi parti de leurs capacités symboliques pour mieux gérer les calculs numériques en cas d'instabilité.

Une étude numérique vise à obtenir les valeurs d'une solution en un certain nombre de points, ou un dessin représentant une ou plusieurs solutions. L'équation ou le système d'équations sera généralement accompagné de conditions *initiales* — les valeurs de la solution et de ses dérivées en un point — ou de conditions *aux limites* — les valeurs sont prises en plusieurs points, habituellement aux extrémités du domaine d'étude. Maple ne propose d'aide que pour le premier cas, en utilisant une méthode de Runge-Kutta par défaut. Il est également possible de changer de méthode, ou de programmer sa propre méthode et de la faire utiliser par le système. Pour plus de détails, nous renvoyons à l'aide en ligne de `dsolve[numeric]`.

Exemple 6. On considère l'équation

$$y' = \sin(xy), \quad y(0) = 1.$$

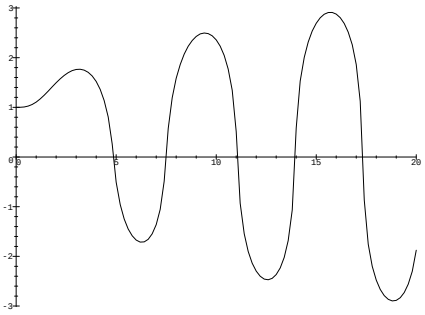


FIGURE 2

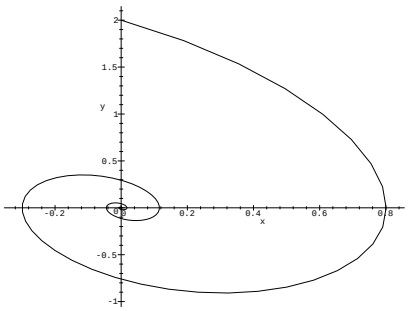


FIGURE 3

D'après le théorème de CAUCHY, l'existence d'une solution et son unicité sont assurées sur tout $\mathbb{R}$. On ne sait pas exprimer la solution à l'aide des fonctions usuelles, mais il est possible d'en calculer des valeurs :

```
f:=dsolve({diff(y(x),x)=sin(x*y(x)),y(0)=1},y(x),numeric):
```

Le résultat est une procédure que l'on peut utiliser pour calculer des valeurs de la solution :

```
f(0.5), f(1);
```

$[x = .5, y(x) = 1.12970783844130440]$, $[x = 1, y(x) = 1.53409997627099437]$

Le tracé de la fonction s'obtient par la commande `DEplot` du *package* `DEtools`.

EXEMPLE 7. Le tracé de la solution de

$$y'(x) = x \sin x \, e^{-|y(x)|}, \qquad y(0) = 1,$$

est obtenu (cf. fig. 2) par :

```
deq:=diff(y(x),x)=x*sin(x)*exp(-abs(y(x))):
DEtools[DEplot](deq,[x,y],0..20,{[0,1]},stepsize=0.2);
```

Comme `DEtools` comporte quelques incorrections, il est bon de savoir que l'on peut également écrire :

```
plots[odeplot](dsolve({deq,y(0)=1},y(x),numeric),y(x),
        0..20,numpoints=100);
```

Diverses options permettent de contrôler l'aspect des courbes obtenues ; plus de détails sont fournis par l'aide en ligne de `DEtools[options]`.

Dans de nombreuses applications physiques, le mouvement d'un point est modélisé par un système d'équations différentielles indiquant comment la dérivée de chacune des coordonnées du point dépend du temps et des autres coordonnées. Nous donnons en figure 3 l'exemple du mouvement plan correspondant au système :

$$\begin{cases} x'(t) = y, \\ y'(t) = -y - 3 \sin x. \end{cases}$$

Le tracé des trajectoires (qui dépendent de la position initiale, ici $x(0) = 0$ et $y(0) = 2$) est obtenu par la commande[†]

```
with(DEtools):
phaseportrait([y,-y-3*sin(x)],[x,y],0..40,
       {[0,0,2]},stepsize=0.1);
```

### 1.4. Exercices.

1. On souhaite construire un échangeur d'autoroute. On considère pour cela que les conducteurs tournent leur volant à vitesse constante, sans changer la vitesse de leur véhicule. Calculer l'équation paramétrique de la courbe parcourue par le véhicule. Cette courbe s'appelle une clothoïde ou spirale de Cornu. [Indication: la courbure est proportionnelle à l'abscisse curviligne.]
2. Aider Maple à terminer la résolution de
$$(x + yy')^2 = x^2 + y^2, \quad (xy' - y)^2 = x^2 - y^2.$$
3. Résoudre les équations
$$|x|y' + (x - 1)y = x^2, \quad 2y'' + 3y = 5|y|.$$
Noter que cette résolution nécessite une discussion.
4. Chercher les fonctions continues sur $\mathbb{R}$ telles que
$$\forall (x, y) \in \mathbb{R}^2, \quad f(x)f(y) = \int_{x-y}^{x+y} f(t)\, dt.$$
5. Chercher une solution $\mathcal{C}^1$ sur $\mathbb{R}$ de
$$f(0) = 0, \quad f'(x) = f(1/x).$$

## 2. Étude différentielle de courbes

Un grand nombre de problèmes géométriques portent sur les propriétés différentielles des courbes dans le plan ou dans l'espace euclidien, c'est-à-dire dans une large mesure sur l'étude des dérivées du vecteur $\overrightarrow{OM}$ ($M$ étant le point courant de la courbe) par rapport à son abscisse curviligne. En particulier, les problèmes faisant intervenir la courbure, les formules de Frenet ou le trièdre de Frenet nécessitent des calculs de dérivées qui sont facilement réalisables à l'aide d'un système de calcul formel. C'est ce que nous allons montrer à travers un exemple de calcul de développée et un exemple de calcul de géodésique.

[†]On devrait pouvoir écrire directement `DEtools[phaseportrait]`, mais le *package* `DEtools` est mal programmé et nécessite l'emploi de `with`.

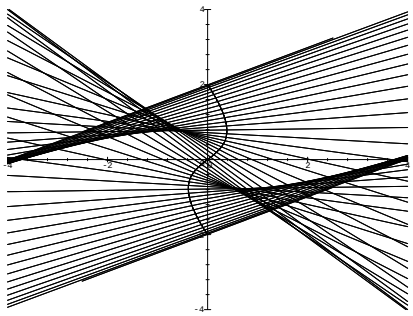


FIGURE 4 La courbe et ses normales.

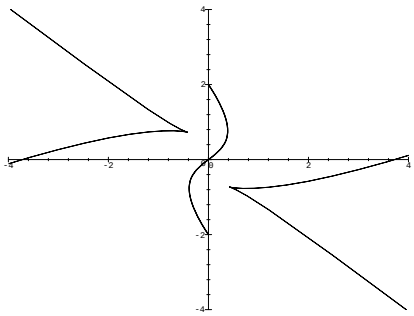


FIGURE 5 La courbe et sa développée.

**2.1. Un calcul de développée.** Pour des courbes dont la représentation paramétrique est compliquée, les calculs de développée sont lourds, mais pas difficiles. Il suivent toujours le même schéma : calculer les équations paramétriques de la famille des normales à la courbe, puis l'enveloppe de cette famille de droites. Nous illustrons cette approche sur un exemple.

EXEMPLE 8. Nous étudions la courbe donnée par la représentation paramétrique

$$\begin{cases} x = \sin^2 t \cos t, \\ y = (1 + \cos^2 t) \cos t. \end{cases}$$

La famille de normales à la courbe s'obtient facilement à partir de ces équations :

```
coords:=[sin(t)^2*cos(t),(1+cos(t)^2)*cos(t)]:
vit:=map(diff,coords,t):
fam:=[coords[1]+vit[2]*lambda,coords[2]-vit[1]*lambda];
```

$$fam := \Big[\sin(t)^2\cos(t) + \big(-2\sin(t)\cos(t)^2 - \big(1+\cos(t)^2\big)\sin(t)\big)\,\lambda,$$
$$\big(1+\cos(t)^2\big)\cos(t) - \big(2\sin(t)\cos(t)^2 - \sin(t)^3\big)\,\lambda\Big]$$

Ces droites ainsi que la courbe de départ sont représentées en figure 4 telles qu'elles ont été obtenues par les commandes :

```
p1:=plot([sin(t)^2*cos(t),(1+cos(t)^2)*cos(t),t=0..2*Pi]):
p2:=plot({seq(subs(t=i*2*Pi/100,[op(fam),lambda=-10..10]),
i=1..100)},-4..4,-4..4):
plots[display]({p1,p2});
```

Pour trouver l'enveloppe, il faut trouver pour chaque normale la valeur de $\lambda$ correspondant au point où la normale est tangente à l'enveloppe. En ce point, la dérivée par rapport au paramètre $t$ doit être parallèle à la normale, ce qui nous donne la valeur de $\lambda$ :

```
map(diff,fam,t):
```

```
simplify(solve(vit[1]*"[1]+vit[2]*"[2],lambda));
```

$$\frac{9\cos(t)^4+1}{6\sin(t)\cos(t)}$$

Il ne reste plus qu'à substituer cette valeur de $\lambda$ dans l'équation des droites :

```
res:=map(simplify,subs(lambda=",fam));
```

$$res := \left[-\frac{27\cos(t)^6-3\cos(t)^2+15\cos(t)^4+1}{6\cos(t)},\right.$$
$$\left.-\frac{-3\cos(t)^2-15\cos(t)^4+27\cos(t)^6-1}{6\cos(t)}\right]$$

La courbe obtenue est tracée sur la figure 5, avec la courbe de départ, par les instructions suivantes :

```
p3:=plot([op(res),t=-Pi/2+0.001..Pi/2-0.001],-4..4,-4..4):
p4:=plot([op(res),t=Pi/2+0.001..3*Pi/2-0.001],-4..4,-4..4):
plots[display]({p1,p3,p4});
```

Dans le cas où la courbe à étudier est beaucoup plus compliquée, on imagine facilement que les calculs pour obtenir la développée peuvent devenir inextricables. Avec un système de calcul formel il suffit de réaliser à nouveau les opérations précédentes en changeant uniquement les équations de départ, c'est-à-dire la liste `coords`.

**2.2. Un calcul de géodésique.** Une géodésique est une courbe d'une surface qui, localement, suit un chemin de longueur minimale entre deux points. Pour de bonnes surfaces, cette propriété est aussi globale. Le calcul se mène simplement en utilisant la propriété que l'accélération d'un point se déplaçant sur la géodésique est en tout point perpendiculaire à la surface.

Exemple 9. Nous allons calculer des géodésiques d'un tore, fourni par sa représentation paramétrique :

```
pt:=[(3+cos(phi))*cos(theta),(3+cos(phi))*sin(theta),sin(phi)];
```

$$pt := [(3+\cos(\phi))\cos(\theta),(3+\cos(\phi))\sin(\theta),\sin(\phi)]$$

Le mouvement d'un point sur la surface est déterminé par la variation des paramètres en fonction du temps :

```
time_dep:=[theta=theta(t),phi=phi(t)]:
```

Le vecteur accélération s'obtient comme dérivée seconde des coordonnées du point par rapport au temps :

```
acc:=diff(subs(time_dep,pt),t,t):
```

Les vecteurs tangents à la surface s'obtiennent par dérivation par rapport aux paramètres :

```
tang:=subs(time_dep,[diff(pt,phi),diff(pt,theta)]):
```

Il ne reste plus qu'à écrire que l'accélération est perpendiculaire à ces deux vecteurs, soit deux produits scalaires :

```
sys:={seq(v[1]*acc[1]+v[2]*acc[2]+v[3]*acc[3],v=tang)}:
```

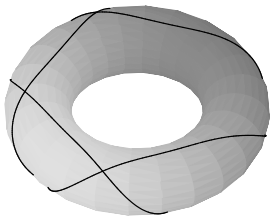


FIGURE 6 Une portion de géodésique du tore.

Ces équations fournissent un système différentiel non linéaire définissant $\phi(t)$ et $\theta(t)$. Dans notre exemple, ces équations se simplifient un peu :

```
combine(sys,trig);
```

$$\Big\{\frac{1}{2}\left(\frac{d}{dt}\theta(t)\right)^2\sin(2\phi(t))+3\sin(\phi(t))\left(\frac{d}{dt}\theta(t)\right)^2+\frac{d^2}{dt^2}\phi(t),$$
$$-6\sin(\phi(t))\frac{d}{dt}\phi(t)\frac{d}{dt}\theta(t)+\frac{19}{2}\frac{d^2}{dt^2}\theta(t)-\frac{d}{dt}\phi(t)\frac{d}{dt}\theta(t)\sin(2\phi(t))$$
$$+6\frac{d^2}{dt^2}\theta(t)\cos(\phi(t))+\frac{1}{2}\frac{d^2}{dt^2}\theta(t)\cos(2\phi(t))\Big\}$$

Maple ne trouve pas de solution formelle à ce système, mais il peut en calculer des solutions numériques :

```
res:=dsolve(sys union {phi(0)=0,theta(0)=0,D(theta)(0)=1,
   D(phi)(0)=4},{theta(t),phi(t)},numeric):
```

Si `plots[odeplot]` fonctionnait correctement, il suffirait de faire

```
plots[odeplot](res,subs(time_dep,pt),0..10);
```

pour tracer une géodésique. Dans la version actuelle, il faut quelques acrobaties supplémentaires :

```
f := t -> eval(subs(time_dep,res(t),pt)):
plots[spacecurve]([seq(f(i/100),i=1..1000)]);
```

Le résultat est représenté en figure 6 simultanément avec le tore.

### 2.3. Exercices.

1. Faire l'étude de la fonction $f(x)=\sqrt{|x-1|}-\ln|x|$ : tableau de variation, concavité et recherche des points d'inflexion, tracé de la courbe.
2. Déterminer la développée de la courbe définie par
$$x(t)=\sin 2t,\quad y(t)=\cos 2t-\ln|\tan t|.$$
3. Faire dessiner au système une géodésique d'une sphère.

CHAPTER XI

# Calculs en Probabilité

Le calcul des probabilités a pour but de modéliser le hasard. Accidents de la circulation, chute des boules au loto, désintégrations de particules, sont des événements apparemment imprévisibles. Cependant, des modèles probabilistes de ces événements permettent aux assurances de calculer leurs tarifs, à l'État d'être sûr de gagner de l'argent sur le loto et aux physiciens de dater les vestiges préhistoriques.

Le calcul formel ne propose pas d'outils spécifiques pour le calcul des probabilités, mais certaines des techniques présentées aux chapitres précédents peuvent y être appliquées avec succès. Ce chapitre est donc essentiellement un chapitre d'application.

Ainsi, les calculs faisant intervenir des variables aléatoires discrètes sont rendus précis par l'utilisation de rationnels exacts et sont simplifiés par les manipulations de séries. Nous aborderons ces aspects en §1 et détaillerons un exemple en §2. L'algèbre linéaire joue également un grand rôle, notamment dans l'étude des chaînes de Markov, mais cette question sort du cadre de ce livre. En §3, nous décrivons quelques techniques de simulation, domaine où, malgré sa lenteur, le calcul formel est attractif par son interactivité. Lorsque la variable aléatoire est continue, les calculs utilisent des intégrales, mais les capacités des systèmes à calculer des primitives ou des intégrales définies ne sont utilisables que sur des exemples simples, aussi nous n'aborderons pas les variables aléatoires continues dans ce chapitre.

## 1. Opérations élémentaires

La figure 1 p. 258 représente le résultat de 500 tirages à pile ou face (voir le §3.1 pour la façon de l'obtenir). En ordonnée sont portés les rapports du nombre de piles obtenus sur le nombre de tirages. La fréquence tend visiblement vers la probabilité 1/2. Cependant quelques questions plus précises peuvent être abordées :

(1) est-il "normal" que la fréquence vaille près de 0,8 après 20 tirages ?
(2) avec quelle probabilité la fréquence reste-t-elle toujours au-dessus de 1/2 ?
(3) quelle est la probabilité que la fréquence soit comprise entre 0,4 et 0,6, ou entre 0,45 et 0,55 après 500 tirages ?

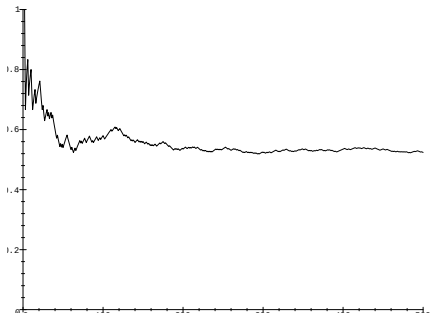


FIGURE 1 Évolution de la proportion du nombre de piles dans un tirage à pile ou face.

**1.1. Probabilités combinatoires.** Lorsque les événements possibles sont en nombre fini et que l'on suppose qu'ils ont tous la même probabilité d'apparition (équiprobabilité), la probabilité d'un événement se réduit au quotient du nombre de cas où cet événement se produit par le nombre total de cas. Ainsi, la probabilité d'avoir pile lors d'un tirage à pile ou face vaut 1/2. Un système de calcul formel est utile dans ce domaine par sa facilité à manipuler des nombres rationnels exacts.

EXEMPLE 1. La probabilité de gagner le gros lot du loto est l'inverse du nombre de tirages possibles, sachant qu'un tirage consiste à tirer 6 boules parmi 49. D'où la probabilité

```
1/binomial(49,6);
```

$$\frac{1}{13983816}$$

ou numériquement :

```
evalf(");
```

$$0.7151123842\,10^{-7}$$

En pratique, le calcul du nombre de cas n'est pas toujours aussi facile, mais les outils introduits au chapitre IV peuvent souvent faciliter la tâche.

**1.2. Sommes de variables aléatoires.** Revenons sur le tirage à pile ou face. La probabilité $p_{n,k}$ d'avoir $k$ piles après $n$ tirages permet d'étudier la fréquence de piles dans la figure 1. Au départ $p_{0,0} = 1$ et $p_{0,k} = 0$ pour $k \neq 0$. Pour arriver à $k$ piles en $n$ tirages, soit on part de $k-1$ piles en $n-1$ tirages et on tire pile, soit on part de $k$ piles en $n-1$ tirages et on tire face. Ceci conduit à la récurrence suivante :

$$p_{n,k} = \frac{1}{2}p_{n-1,k-1} + \frac{1}{2}p_{n-1,k}.$$

En oubliant les facteurs 1/2, il s'agit là de la récurrence des coefficients binomiaux du triangle de PASCAL. Ce n'est pas étonnant puisque pour obtenir $k$ piles en $n$ tirages, il suffit de choisir les positions de $k$ tirages donnant pile parmi $n$

et les $n-k$ tirages qui restent donnent forcément face, d'où $p_{n,k} = \binom{n}{k}2^{-n}$. Une autre façon de le voir consiste à utiliser des séries génératrices.

1.2.1. *Séries génératrices de probabilités*

La série génératrice de probabilité associée à une variable aléatoire discrète $X$ est par définition

$$f(z) = \sum_k \Pr[X = k] z^k \tag{1}$$

où $\Pr[X = k]$ est la probabilité d'apparition de la valeur $k$ et où la sommation vaut sur toutes les valeurs possibles de la variable aléatoire $X$. En conséquence $f(1) = 1$ et le rayon de convergence vaut au moins 1. La suite des coefficients s'appelle la *distribution de probabilité* de la variable $X$.

Exemple 2. Dans le cas du jet d'une pièce, la variable aléatoire étant le nombre d'occurrences de pile, elle ne peut prendre pour valeur que 0 ou 1, chacune avec probabilité 1/2. La série génératrice est donc le polynôme $1/2 + z/2$.

```
f := 1/2+z/2:
```

L'intérêt des séries génératrices vient de ce que la série génératrice associée à la somme de deux variables aléatoires indépendantes s'obtient en multipliant leurs séries respectives. Pour s'en convaincre, il suffit de regarder ce qui se passe au niveau des monômes. Le produit de $a_i z^i$ par $b_j z^j$ donne $a_i b_j z^{i+j}$ ; les probabilités $a_i, b_j$ sont bien multipliées grâce à l'hypothèse d'indépendance et les occurrences $i, j$ additionnées.

Exemple 3. Ceci nous permet de répondre à deux des questions posées ci-dessus. La série associée à une suite de 20 tirages est $f^{20}$ ; la probabilité d'avoir au moins 80 % de piles est la somme des coefficients de $z^{16}$, $z^{17}$, $z^{18}$, $z^{19}$ et $z^{20}$.

```
f20 := expand(f^20):
convert([seq(coeff(f20,z,j),j=16..20)],`+`);
```

$$\frac{1549}{262144}$$

Pour 500 tirages, la probabilité d'avoir une fréquence comprise entre 0,4 et 0,6 correspond à un nombre d'occurrences de pile entre 200 et 300, et de même 0,45 et 0,55 correspondent à 225 et 275.

```
f500 := expand(f^500):
evalf(convert([seq(coeff(f500,z,j),j=200..300)],`+`)),
evalf(convert([seq(coeff(f500,z,j),j=225..275)],`+`));
```

$$0.9999941484, \quad 0.9775337951$$

Autrement dit, si l'on prédit qu'après 500 tirages le nombre de piles obtenus est compris entre 200 et 300, la probabilité de se tromper vaut moins de 0,0006 %, alors que si l'on prédit qu'il est compris entre 225 et 275, on a tort avec une probabilité proche de 2 %.

Exemple 4. Dans l'exemple précédent, toutes les probabilités pouvaient s'obtenir comme des coefficients binomiaux. Ce n'est plus le cas lorsque l'on considère les dés, en particulier pour la somme des valeurs lors de $k$ lancers. Au bout de 100 jets de dés la probabilité pour que la somme soit comprise entre 300 et 400 s'obtient par :

```
de100:=expand((z/6+z^2/6+z^3/6+z^4/6+z^5/6+z^6/6)^100):
evalf(convert([seq(coeff(de100,z,j),j=300..400)],`+`));
```

0.9969883792

1.2.2. *Espérance et écart type*

Lorsque des valeurs numériques sont associées aux événements possibles (lancer d'un dé par exemple), on définit l'espérance mathématique qui correspond à la limite de la moyenne arithmétique des valeurs observées. C'est une information importante lorsque par exemple la variable aléatoire représente un gain dans une partie. L'espérance représente alors le gain espéré et il faut faire en sorte de la maximiser.

Si $X$ est une variable aléatoire discrète, son espérance est

$$E(X) = \sum_k k \Pr[X = k] = p'(1),$$

où $p$ est la série génératrice de probabilité. Cette espérance n'est pas nécessairement finie, mais elle l'est pour beaucoup d'exemples courants. Nous verrons en §2.2 un exemple de variable dont l'espérance n'est pas finie.

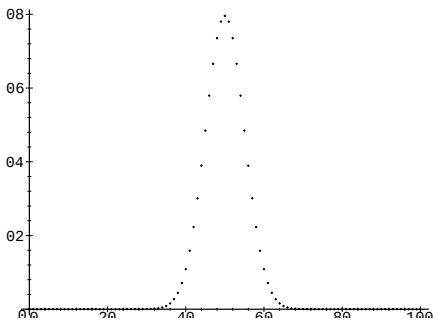
L'espérance de l'écart à $E(X)$ est naturellement nulle. La *variance*, qui est l'espérance du carré de l'écart à $E(X)$, précise à quel point une distribution est *centrée* sur son espérance, c'est-à-dire à quel point les valeurs éloignées de l'espérance sont probables. Là encore, la connaissance de la fonction génératrice facilite le calcul :

$$\begin{aligned} V(X) &= \sum_k [E(X) - k]^2 \Pr[X = k] \\ &= E(X)^2 - 2E(X)^2 + \sum_k k^2 \Pr[X = k] \\ &= p''(1) + p'(1) - p'(1)^2. \end{aligned}$$

Si la variable aléatoire représente une quantité physique dans une certaine unité, la racine carrée de la variance, appelée *écart-type*, est dans la même unité.

1.2.3. *Moments d'ordre supérieur*

Espérance et variance sont précisées par les *moments* de la variable aléatoire. Le moment d'ordre $n$ est par définition $E(X^n)$ et le moment centré $\mu_n = E((X - E(X))^n)$ s'en déduit par linéarité de l'espérance. D'après la section précédente, la variance $\mu_2 = V(X)$ est reliée aux dérivées de la fonction génératrice et il en va de même pour les autres moments. Le lien s'obtient




```
f100 := expand((1/2+z/2)^100):
plot([seq([i,coeff(f100,z,i)],i=0..100)],style=point);
```

FIGURE 2 Distribution du nombre de piles lors de 100 tirages.

grâce aux nombres de STIRLING de seconde espèce $S_{n,j}$ (`combinat[stirling2]` en Maple) que l'on peut définir par :

$$k^n = \sum_{j=1}^{n} S_{n,j} k(k-1)\cdots(k-j+1),$$

d'où

$$E(X^n) = \sum_{j=1}^{n} S_{n,j} p^{(j)}(1).$$

Le quatrième moment est ainsi

```
E(X^4)=convert([seq(stirling2(4,j)*(D@@j)(p)(1),j=1..4)],`+`);
```

$$E(X^4) = \mathrm{D}(p)(1) + 7D^{(2)}(p)(1) + 6D^{(3)}(p)(1) + D^{(4)}(p)(1)$$

qui se lit $p'(1) + 7p''(1) + 6p'''(1) + p^{(4)}(1)$.

1.2.4. *Distributions classiques*

Dans les exemples les plus simples qui viennent à l'esprit (tirage à pile ou face, lancer de dé, tirage du loto), tous les événements ont la même probabilité de se produire. La distribution de probabilité est alors *uniforme*.

Mais dans de nombreux cas courants, la distribution n'est pas uniforme. Il en va ainsi pour le nombre de piles lors de $k$ tirages, dont la distribution de probabilités est tracée en figure 2 pour $k = 100$. C'est un exemple de distribution binomiale. Plus généralement, si $p$ est la probabilité d'un certain événement et $q = 1 - p$, la probabilité qu'il se réalise $k$ fois lors de $n$ tirages indépendants est le coefficient de $z^k$ dans $(pz + q)^n$, c'est-à-dire

$$\binom{n}{k} p^k q^{n-k}.$$

Espérance et variance du nombre de réalisations s'obtiennent facilement :

```
f[0]:=(p*z+1-p)^n: f[1]:=diff(f[0],z): f[2]:=diff(f[1],z):
subs(z=1,f[1]), factor(subs(z=1,f[2]+f[1]-f[1]^2));
```

$$np, \quad np(1-p)$$

L'espérance est conforme à l'intuition : après $n$ tirages où l'événement a une probabilité $p$, on espère l'avoir observé $np$ fois. Quant à la variance, elle croît linéairement avec $n$, ce qui signifie que l'écart-type ne croît qu'en $\sqrt{n}$. Donc lorsque $n$ croît, les valeurs (après division par $n$) se regroupent autour de l'espérance $p$. On peut même être beaucoup plus précis : lorsque $n$ tend vers l'infini, la probabilité que la valeur soit comprise entre $np-\alpha\sqrt{np(1-p)}$ et $np+\alpha\sqrt{np(1-p)}$ tend vers

$$\frac{1}{\sqrt{2\pi}}\int_{-\alpha}^{\alpha} e^{-x^2/2}dx. \tag{2}$$

Cette approximation s'appelle traditionnellement l'approximation gaussienne, bien qu'elle ait été utilisée avant Gauss par De Moivre et Laplace.

Exemple 5. Pour calculer à l'exemple 3 la probabilité d'avoir une fréquence de piles comprise entre 0,4 et 0,6 au bout de 500 tirages, il avait fallu développer un polynôme de degré 500. Si l'on augmente trop ce degré, le calcul devient impraticable. En revanche, l'approximation prend toujours le même temps de calcul :

```
N:=expand(1/sqrt(2*Pi)*int(exp(-x^2/2),x=-alpha..alpha));
```

$$N := \operatorname{erf}\left(\frac{\alpha}{\sqrt{2}}\right)$$

```
seq(evalf(subs(alpha=d/sqrt(500/4),N)),d=[50,25]);
```

$$0.9999922558, \quad 0.9746526813$$

Inversement, pour une probabilité donnée, l'expression de $N$ permet de trouver la valeur de $\alpha$ donnant cette probabilité dans (2).

```
fsolve(N=0.95), fsolve(N=0.99);
```

$$1.959963985, \quad 2.575829304$$

Par conséquent, 95% des valeurs sont dans un intervalle de largeur 1,96 fois l'écart-type autour de la moyenne et 99% dans un intervalle de largeur 2,58 fois l'écart-type.

La distribution binomiale se généralise en distribution multinomiale lorsque plus de deux cas sont possibles à chaque tirage. En effet, si $p_1+\cdots+p_r=1$ sont les probabilités d'événements disjoints $A_1,\ldots,A_r$, alors la probabilité que lors de $n$ épreuves, $A_1$ se produise $k_1$ fois,..., $A_r$ se produise $k_r$ fois est le coefficient de $x_1^{k_1}\cdots x_r^{k_r}$ dans le polynôme

$$(p_1x_1+\cdots+p_rx_r)^n.$$

Il vaut généralement mieux calculer avec des séries car ces polynômes ont un grand nombre de coefficients.

Par exemple, la probabilité que lors de 20 lancers de dés, le deux apparaisse 4 fois et le six 3 fois s'obtient par :

```
p:=(x[2]/6+x[6]/6+4/6)^20:
coeff(series(coeff(series(p,x[2],5),x[2],4),x[6],4),x[6],3);
```

$$\frac{57881600}{1162261467}$$

*Distribution de* Poisson. La distribution de Poisson a pour fonction génératrice

$$e^{\lambda(z-1)} = e^{-\lambda} \sum_{n=0}^{\infty} \frac{\lambda^n}{n!} z^n.$$

Elle s'obtient comme limite de la distribution binomiale lorsque la probabilité $p = \lambda/n$ de l'événement observé est faible par rapport au nombre d'observations $n$.

Exemple 6. Si la probabilité d'une faute de frappe vaut 0,0012 par caractère, la probabilité qu'une page de 2 500 caractères contienne zéro, une ou deux fautes vaut respectivement

```
p:=0.0012:
(1-p)^2500, 2500*p*(1-p)^2499, 2500*2499/2*p^2*(1-p)^2498;
```

0.04969746062, 0.1492715077, 0.2240866026

et l'approximation de Poisson donne :

```
lambda:=p*2500:
exp(-lambda), exp(-lambda)*lambda, exp(-lambda)*lambda^2/2;
```

0.04978706837, 0.1493612051, 0.2240418077

La fonction génératrice donne l'espérance et la variance :

```
f[0]:=exp(lambda*(z-1)):f[1]:=diff(f[0],z):f[2]:=diff(f[1],z):
eval(subs(z=1,[f[1], f[2]+f[1]-f[1]^2]));
```

$$[\lambda, \quad \lambda]$$

**1.3. Produits de variables aléatoires.** La définition des séries génératrices de probabilité donnée plus haut est surtout adaptée à l'étude de la *somme* de variables aléatoires indépendantes. Dans le cas du produit, il faut utiliser une série de Dirichlet de la forme

$$f(s) = \sum_k \Pr[X = k] \frac{1}{k^s}.$$

En effet, dans le produit de $1/i^s$ par $1/j^s$, les valeurs $i$ et $j$ se multiplient pour donner $1/(ij)^s$. Ici c'est le nombre élevé à la puissance $s$ qui permet de "suivre" la valeur de la variable.

Exemple 7. Avec quelle probabilité le produit des valeurs de cinq dés est-il compris entre 100 et 1 000 ? Les séries de Dirichlet associées à un et à cinq tirages valent

```
f := convert([seq(1/6/j^s,j=1..6)],`+`):
f5 := expand(f^5):
```

Il ne reste plus qu'à garder les termes de l'intervalle considéré.

```
test := a -> evalb(100<=a and a<=1000):
f5bon := select(t->test(limit(t^(-1/s),s=infinity)),f5):
```

Le remplacement de $s$ par 0 élimine les termes en $k^s$ et donne la probabilité cherchée.

```
eval(subs(s=0,f5bon));
```

$$\frac{4631}{7776}$$

### 1.4. Exercices.

1. Problème du chevalier de Méré. Des deux événements suivants, quel est le plus probable ?
   (1) Obtenir au moins une fois 6 en lançant un dé 4 fois ;
   (2) obtenir au moins un double 6 en lançant deux dés 24 fois.
2. Après 10 lancers de dés, avec quelle probabilité la somme des points fait-elle $k$ ? Tracer la courbe. Recommencer avec 100 lancers.
3. Calculer la probabilité d'avoir trois bons numéros au loto.
4. La queue du cinéma contient 100 personnes. La caisse est vide. Le billet coûte 50 F. Dans la queue, 40 personnes n'ont que des billets de 100 F et 60 n'ont que des billets de 50 F. Quelle est la probabilité qu'aucune personne ne se présente à la caisse sans que l'on puisse lui rendre la monnaie ?
5. Huit tours sont placées sur un échiquier. Quelle est la probabilité qu'aucune des tours ne puisse prendre une autre ?
6. Sur 1 000 oranges à emballer, 10 sont pourries. Ces oranges sont mises par sacs de 8. Combien peut-on espérer de sacs vendables ? Comparer avec l'approximation de Poisson.
7. Le paradoxe de Saint-Petersbourg. Un homme joue à pile ou face avec de l'argent : si la face sur laquelle il a parié sort, il gagne autant que sa mise, et la perd sinon. Sa stratégie consiste à s'arrêter dès qu'il a gagné, et à doubler sa mise en cas de défaite. En démarrant avec une mise de 1 F, quelle est son espérance de gain ? Que devient cette espérance si sa fortune est limitée à 16 384 F ? Quelle est la probabilité que cette stratégie paye $k$ parties successives (en repartant toujours avec la même somme initiale) ?
8. Le Tapis Vert. Dans ce jeu quotidien créé en 1987, on choisit dans un jeu de 32 cartes une carte de chaque couleur (trèfle, pique, carreau, cœur). Les mises étant faites, une combinaison de quatre cartes est tirée au hasard. Avec deux cartes exactes, on gagne 2 fois la mise, avec trois cartes 30 fois, et avec quatre cartes 1 000 fois la mise.
   (1) Calculer l'espérance de gain pour une mise de un franc.
   (2) Le 29 mars 1988, le carré d'as est sorti ; 22 000 joueurs avaient choisi cette combinaison et la Française des Jeux a dû débourser 100 fois le total des mises ce jour là. Quelle était la probabilité que le carré d'as sorte au moins une fois dans les 165 premiers jours après la création du jeu ? Quelle est l'espérance du nombre de jours entre deux sorties du carré d'as ?

9. Le paradoxe des anniversaires. En supposant les 365 jours de l'année équiprobables, calculer à partir de combien de personnes dans une assemblée la probabilité qu'au moins deux d'entre elles aient le même jour anniversaire est supérieure à 1/2.

   Même question pour 3 personnes avec le même anniversaire.

## 2. Marches aléatoires et problèmes de ruine

Le modèle des marches aléatoires s'applique à de nombreuses quantités qui évoluent par transitions aléatoires. Il nous permet d'illustrer l'utilisation des séries génératrices et d'obtenir plusieurs exemples de résultats contraires à l'intuition.

**2.1. Règles du jeu et problèmes.** Le problème de la ruine se présente sous la forme d'un jeu : à chaque étape une pièce est lancée. Si pile sort, le premier joueur reçoit un franc de son adversaire, sinon il lui donne un franc. Pour rendre le modèle plus général, on considère une pièce imparfaite donnant pile avec probabilité $p$ (on note à nouveau $q = 1 - p$).

Le jeu dure $n$ étapes. Dans un premier temps la fortune des deux joueurs est supposée illimitée. Il s'agit de répondre aux questions suivantes :

(1) au bout de combien d'étapes les deux joueurs reviennent-ils à égalité pour la première fois ?
(2) quelle est l'espérance de gain de chacun des joueurs ?
(3) combien de fois les joueurs sont-ils à égalité au cours de la partie ?
(4) au bout de combien d'étapes les deux joueurs reviennent-ils à égalité pour la dernière fois ?
(5) en supposant la fortune d'un joueur limitée à $k$, quelle est la probabilité qu'il ait tout perdu en $n$ étapes ? Combien d'étapes dure en moyenne la partie ?

Une représentation graphique d'évolution du gain du premier joueur est donnée en figure 3 p. 266. C'est une marche aléatoire d'un point partant de l'origine, dont l'abscisse augmente d'une unité à chaque étape, et dont l'ordonnée augmente d'une unité lorsque pile sort et décroît d'une unité sinon. Les deux joueurs sont à égalité lorsque le point retourne sur l'axe des abscisses. La limitation sur les fortunes revient à rajouter des barrières horizontales. Lorsque le point arrive sur une telle barrière, il est absorbé, ce qui correspond à la ruine d'un joueur, et à la fin de la partie.

Les questions (1), (2) et (5) sont traitées dans les sections qui suivent ; des indications sur les autres questions sont données dans les exercices.

**2.2. Premier retour.** Si $Y$ est l'abscisse du premier retour à une ordonnée nulle (égalité des joueurs), la fonction génératrice de $Y$ s'obtient en considérant la structure de "l'arche" que doit parcourir le point : cette arche commence par un pas vers le haut ou vers le bas, se termine par le pas inverse, et entre les deux le point peut parcourir une succession d'arches définies de la

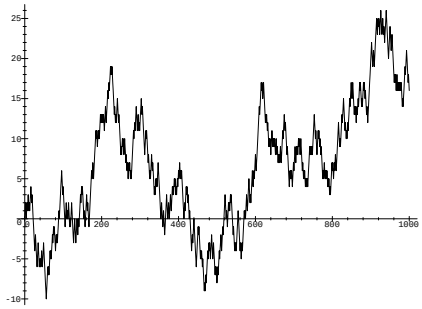



FIGURE 3 Évolution de la fortune des joueurs (ici $p = q = 1/2$).

même façon. Cette définition se traduit en termes de fonctions génératrices, par exemple pour les arches montantes :

$$A(z) = pz\frac{1}{1 - A(z)}qz.$$

Le facteur $pz$ est la fonction génératrice d'un pas vers le haut. Il est suivi d'une suite $1/(1 - A(z))$ (voir p. 108) d'arches indépendantes de fonction génératrice $A(z)$, et enfin d'un pas vers le bas, de fonction génératrice $qz$. Cette équation se résout :

```
solve(A=p*q*z^2/(1-A),A);
```

$$\frac{1}{2} + \frac{1}{2}\sqrt{1 - 4pqz^2}, \quad \frac{1}{2} - \frac{1}{2}\sqrt{1 - 4pqz^2}$$

La première solution ayant des coefficients de TAYLOR négatifs, c'est la seconde qui nous intéresse :

```
A:="[2]:
```

Les arches descendantes vérifiant la même équation, elles ont la même fonction génératrice. La probabilité que le premier retour ait lieu à l'abscisse $2n$ est le coefficient de $z^{2n}$ dans $2A(z)$, et la probabilité de retour est la somme de ces coefficients, c'est-à-dire $2A(1)$ :

```
2*subs(z=1,A);
```

$$1 - \sqrt{1 - 4pq}$$

Le seul cas où le retour est certain est par conséquent $p = q = 1/2$. En cas de retour, l'espérance de l'abscisse du premier retour vaut :

```
2*subs(z=1,diff(A,z));
```

$$4\frac{pq}{\sqrt{1 - 4pq}}$$

Lorsque $p \neq 1/2$ le retour n'est donc pas certain, mais lorsqu'il se produit sa position a une espérance finie, alors que lorsque $p = 1/2$, le retour est certain, mais sa position a une espérance infinie. La variance s'obtient de la même manière.

Après qu'une arche a été parcourue, la situation est revenue au point de départ, et la suite est indépendante de ce qui s'est passé pendant la première

arche. La fonction génératrice du $k^{\text{e}}$ retour à l'origine vaut donc $[2A(z)]^k$ et dans la fonction génératrice

$$B(z) = \frac{1}{1-2A(z)},$$

le coefficient de $z^n$ est la probabilité qu'un retour ait lieu à l'abscisse $n$.

**2.3. Gain.** Après avoir parcouru un certain nombre d'arches, le point "décolle" et quitte l'axe des abscisses pour ne plus y revenir. S'il part vers le haut (resp. le bas), sa trajectoire se décompose en une succession d'arches suivie d'un pas montant (resp. descendant), suivi d'une succession d'arches montantes (resp. descendantes), suivie d'un pas montant (resp. descendant)... D'où les deux fonctions génératrices

$$C^+(z) = \frac{B(z)}{1-\frac{pz}{1-A(z)}}, \quad C^-(z) = \frac{B(z)}{1-\frac{qz}{1-A(z)}}, \tag{3}$$

la première correspondant aux parties gagnées par le premier joueur, et la seconde à celles qu'il perd. Ces formules traduisent la décomposition ci-dessus : $B(z)$ correspond à la succession d'arches montantes ou descendantes parcourues jusqu'au dernier départ de l'axe des abscisses, puis, indépendamment de ce qui s'est passé auparavant, le point parcourt une succession de quasi-arches, chacune étant formée d'un pas montant $(pz)$ suivi d'une successions d'arches montantes $(1/(1-A(z)))$. Le coefficient de $z^n$ dans $C^+(z)$ (resp. $C^-(z)$) est donc la probabilité que l'ordonnée à l'abscisse $n$ soit positive (resp. négative).

```
Cp:=1/(1-2*A)/(1-p*z/(1-A)):
```

$$Cp := \frac{1}{\sqrt{1-4pqz^2}\left(1-\frac{pz}{\frac{1}{2}+\frac{\sqrt{1-4pqz^2}}{2}}\right)}$$

Les premières valeurs de ces probabilités sont

```
map(expand,series(Cp,z));
```

$$1 + pz + (2pq+p^2)z^2 + (3p^2q+p^3)z^3 + (6p^2q^2+4p^3q+p^4)z^4 + (10p^3q^2+5p^4q+p^5)z^5 + \mathrm{O}(z^6)$$

Chacun de ces coefficients s'interprète facilement. Par exemple pour que le point ait une ordonnée positive à la troisième étape, il faut que les tirages aient été pile-face-pile, face-pile-pile, pile-pile-face ou pile-pile-pile. La somme des probabilités de ces trois événements donne le coefficient de $z^3$.

Le comportement asymptotique de cette probabilité (la victoire du premier joueur) lorsque le nombre d'étapes tend vers l'infini s'étudie comme on l'a vu au chapitre VIII. La première phase consiste à chercher la singularité la plus proche de l'origine. La fonction $A(z)$ est singulière en $\rho = 1/\sqrt{4pq}$, où elle vaut $1/2$. La position relative de $2p\rho = \sqrt{p/q}$ et de 1 détermine alors la singularité dominante.

Si $2p\rho < 1$, c'est-à-dire si $p < 1/2$, les singularités dominantes sont $\rho$ et $-\rho$, où $1 - 4pqz^2$ s'annule. Le comportement au voisinage de $\rho$ est le suivant :

```
series(subs(z=1/sqrt(4*p*q)*(1-u),Cp),u,2);
```

$$\frac{1}{2}\frac{\sqrt{2}}{\left(1-\frac{p}{\sqrt{pq}}\right)\sqrt{u}} + \frac{p}{\left(-\sqrt{pq}+p\right)\left(1-\frac{p}{\sqrt{pq}}\right)} + \mathrm{O}\left(\sqrt{u}\right)$$

et celui au voisinage de $-\rho$ est similaire. L'équation (9) page 214 donne pour le coefficient de $(1-z/\rho)^{-1/2}$ l'équivalent $\rho^{-n}/\sqrt{\pi n}$. En tenant compte des deux contributions, en $\rho$ et $-\rho$, le premier terme du développement asymptotique de la probabilité de victoire du premier joueur pour un jeu en $n$ étapes est pour $n$ pair

$$\frac{q\sqrt{2}}{(q-p)\sqrt{\pi}}(4pq)^{n/2}n^{-1/2}.$$

Pour $n$ impair seule la constante diffère, la contribution en $-\rho$ venant se retrancher au lieu de s'ajouter. Cette probabilité décroît exponentiellement avec $n$ car $4pq < 1$.

Si $2p\rho = 1$, c'est-à-dire si $p = 1/2$, la singularité dominante est toujours en $\rho = 1$, mais le comportement au voisinage de la singularité a changé :

```
series(subs(p=1/2,q=1/2,Cp),z=1,2);
```

$$-\frac{1}{2}\frac{1}{z-1} - \frac{1}{4}\frac{I\sqrt{2}}{\sqrt{z-1}} + \mathrm{O}(1)$$

d'où le comportement asymptotique

$$\frac{1}{2} + O(1/\sqrt{n})$$

qui correspond bien à la probabilité 1/2 qu'a chacun des joueurs de gagner dans ce cas.

Si $2p\rho > 1$, c'est-à-dire si $p > 1/2$, la singularité dominante est atteinte lorsque $pz = 1 - A(z)$, c'est-à-dire en $z = 1$. Le comportement singulier est alors donné par :

```
assume(p>1/2);
map(normal,series(subs(q=1-p,Cp),z=1,2));
```

$$-(z-1)^{-1} + O(1)$$

d'où l'on déduit le comportement asymptotique

$$1 + O(1/n).$$

Le comportement asymptotique dû au pôle suivant donne un reste exponentiellement faible : c'est par symétrie la probabilité de gagner lorsque $p < 1/2$. Le comportement dans ce cas est conforme à l'intuition : dès que $p > 1/2$ la probabilité de gagner tend vers 1 lorsque la durée de la partie tend vers l'infini. L'espérance de gain fait l'objet de l'exercice 1.

**2.4. Ruine.** On suppose que la fortune du premier joueur est limitée (il joue contre un casino, ou il s'agit d'une compagnie d'assurance,...). La série génératrice associée à la probabilité de ruine à l'étape $n$ s'obtient en décomposant la partie comme précédemment : au départ le point parcourt un certain nombre d'arches montantes, puis une étape vers le bas, puis des arches montantes,... Contrairement au cas précédent, le nombre d'étapes vers le bas est limité par la fortune du joueur. Si $k$ est cette fortune, la fonction génératrice s'écrit donc :

$$\left(\frac{qz}{1-A(z)}\right)^k$$

et l'espérance du nombre d'étapes avant la ruine vaut :

```
p:='p':
normal(subs(z=1,diff((q*z/(1-A))^k,z)));
```

$$\frac{k\left(2\dfrac{q}{1+\sqrt{1-4pq}}\right)^k}{\sqrt{1-4pq}}$$

La singularité de ce résultat pour $p = q = 1/2$ montre qu'alors l'espérance du nombre d'étapes est infinie.

**2.5. Exercices.**

1. Le calcul de l'espérance du gain dans la marche aléatoire est simplifié par l'utilisation de fonctions génératrices à deux variables.
   (1) Justifier pourquoi la fonction obtenue en remplaçant $pz$ (resp. $qz$) par $pvz$ (resp. $qvz$) dans l'expression de $C^+(z)$ (resp. $C^-(z)$), donnée p. 267, est une fonction génératrice où le coefficient de $v^k z^n$ est la probabilité qu'à l'abscisse $n$ l'ordonnée vaille $+k$ (resp. $-k$).
   (2) À partir des dérivées par rapport à $v$ en $v = 1$ de ces fonctions, calculer la fonction génératrice des espérances de gain du premier joueur. Vérifier les premiers coefficients.
   (3) En procédant comme pour la probabilité de gagner, calculer le comportement asymptotique de l'espérance de gain.
2. Cet exercice propose d'étudier le nombre de passages par la position de départ pendant la partie.
   (1) Construire une fonction génératrice à deux variables $u$ et $z$ où le coefficient de $z^n u^k$ est la probabilité que l'ordonnée soit positive après $n$ étapes de jeu, le jeu étant passé $k$ fois à l'équilibre. [Il suffit de mettre $u$ au bon endroit dans $C^+$.]
   (2) Comme dans l'exercice précédent, en déduire la fonction génératrice de l'espérance du nombre de passages à l'équilibre.
   (3) Déterminer alors singularité dominante et comportement asymptotique.
3. Cet exercice propose de déterminer l'espérance de la dernière abscisse où l'ordonnée est nulle. L'intuition peut être guidée par les simulations effectuées à l'exercice 1 p. 273.

(1) Déterminer une fonction génératrice à deux variables $z$ et $t$ où le coefficient de $z^n t^m$ est la probabilité que le dernier passage à l'ordonnée 0 précédant l'étape $n$ ait eu lieu à l'étape $m$. [Il suffit de mettre $t$ au bon endroit dans $C^+ + C^-$.]
(2) En déduire la fonction génératrice des espérances cherchées.
(3) Mener l'étude asymptotique.
(4) Étudier la variance.

4. Le premier joueur a une fortune limitée. Il décide que sa stratégie sera de s'arrêter dès qu'il a augmenté sa fortune initiale de $\ell$.
(1) Calculer la fonction génératrice dont le coefficient de $z^n$ est la probabilité pour que le joueur atteigne le but qu'il s'est fixé à l'étape $n$.
(2) Calculer la probabilité qu'il gagne.
(3) Calculer l'espérance de la durée de la partie s'il est ruiné.
(4) Calculer son espérance de gain.
5. Lorsque les deux joueurs ont une fortune limitée différente, calculer en fonction de ces fortunes la probabilité qu'a chacun de ruiner l'autre.

## 3. Simulation

Bien souvent il est impossible de déterminer exactement une distribution ou les caractéristiques d'une variable aléatoire. La simulation permet de faire rapidement de nombreuses expériences, de tester différents modèles, ou de vérifier des résultats théoriques. Nous montrons dans cette section des exemples d'utilisation d'un générateur aléatoire uniforme, et comment écrire un générateur aléatoire selon une distribution donnée.

**3.1. Tirage uniforme.** La figure 1 p. 258 n'a pas été obtenue en tirant 500 fois à pile ou face. Elle est le résultat de l'utilisation de la commande `rand` qui fournit une procédure renvoyant un entier aléatoire uniformément dans un intervalle donné en argument :

```
piece:=rand(2): pile[0]:=0:
for i to 1000 do pile[i]:=pile[i-1]+piece() od:
plot([seq([i,pile[i]/i],i=1..500)],n=0..500,p=0..1);
```

Pour obtenir une distribution, il faut faire plusieurs séries de tirages. Par exemple, la courbe expérimentale correspondant à la distribution théorique de la figure 2 p. 261 s'obtient en comptant le nombre moyen de piles dans une série de 100 tirages. Pour compter le nombre de piles, il suffit d'écrire une procédure qui vaut 1 en cas de pile et 0 sinon :

```
pile:=rand(2):
```

puis on écrit une procédure qui compte le nombre de piles dans une série de 100 tirages :

```
nbpile:=proc() local i;convert([seq(pile(),i=1..100)],`+`) end:
```

ensuite on fait des séries de 100 tirages, et on compte combien de fois le nombre de piles vaut $k$, pour $0 \leq k \leq 100$ :

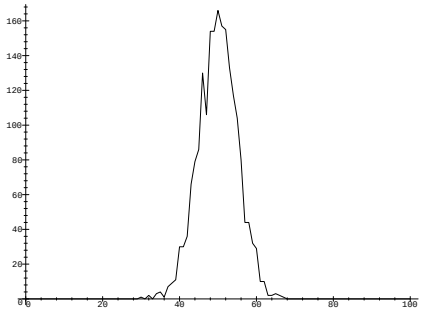



FIGURE 4

```
s:=array(0..100,sparse):
to 2000 do n:=nbpile(); s[n]:=s[n]+1 od:
```

La figure 4 s'en déduit alors par

```
plot([seq([n,s[n]],n=0..100)]);
```

Lorsque la variable aléatoire est continue, il faut lisser la courbe en groupant les résultats proches. Nous en verrons un exemple plus loin.

La fonction `rand` renvoie un générateur d'entiers ; une approximation d'un générateur uniforme sur l'intervalle réel $[0,1]$ s'obtient ainsi :

```
N := 10^20:
randN := rand(0..N-1):
randU := () -> evalf(randN()/N):
```

**3.2. Tirage selon une distribution fixée.** On a souvent besoin de tirer des nombres de manière aléatoire, mais pas uniforme. À partir d'un générateur aléatoire uniforme, il est possible de construire un générateur aléatoire suivant une distribution de probabilité donnée. La distribution est définie par la donnée de l'intervalle $(a,b)$ parcouru par la variable aléatoire $X$, et par sa fonction de répartition $F(t) = \Pr[X \leq t]$. La fonction $F$ transforme l'intervalle $(a,b)$ en $[0,1]$, et il n'est pas difficile de voir que c'est la fonction inverse que nous cherchons : $F^{-1}$ transforme la distribution uniforme sur $[0,1]$ en la distribution désirée sur $(a,b)$. Lorsque $F$ admet un inverse simple, il suffit donc de tirer $X = F^{-1}(U)$ où $U$ est uniforme sur $[0,1]$.

EXEMPLE 8. Un générateur aléatoire selon la loi exponentielle sur $[0,\infty[$, dont la fonction de répartition est $1-\exp(-t)$, s'obtient par

```
rand_exp:=proc() -log(randU()) end:
```

en utilisant le générateur `randU` vu plus haut, et le fait que $U$ et $1-U$ ont la même distribution. Une vérification est obtenue en traçant la courbe, de la même façon que ci-dessus :

```
s:=array(0..100,sparse):
to 2000 do
```

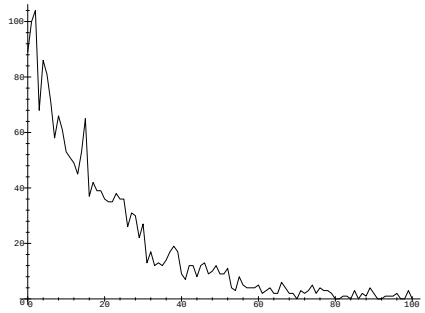



FIGURE 5 Tirages selon une loi exponentielle.

```
    t:=trunc(20*rand_exp());
    if t<=100 then s[t]:=s[t]+1 fi
od;
plot([seq([n,s[n]],n=0..100)]);
```

ce qui donne la figure 5.

Lorsque la fonction de répartition a une expression compliquée, il n'est pas si facile de calculer son inverse. Voici une procédure qui prend en entrée une fonction de répartition $F$ de variable $x$, un intervalle $r$, et retourne un générateur aléatoire (peu efficace) suivant la distribution voulue.

```
gen_rand := proc(F,x,r)
   subs(_corps='fsolve(F=randU(),x=r)',proc() _corps end)
end:
```

Considérons par exemple la distribution de fonction de répartition $(x+\sin x)/\pi$ sur l'intervalle $[0,\pi]$ ;

```
F1 := (x+sin(x))/Pi:
```

le générateur aléatoire correspondant est obtenu par

```
randF1 := gen_rand(F1,x,0..Pi):
```

Voici la procédure que nous avons construite

```
print(randF1);
```

**proc**() fsolve((x+sin(x))/Pi = randU(),x = 0 .. Pi) **end**

et voici quelques tirages:

```
seq(randF1(),i=1..12);
```

.4556513441, 1.030514031, 1.336654437, .6295240942,
.1501246179, .7593796395, .2153416371, .09045415529,
.6006381568, .1712310204, .3114670168, .5318833976

À titre de vérification, l'espérance de la distribution définie ci-dessus est donnée par

```
esp := normal(int(x*diff(F1,x),x=0..Pi));
```

$$esp := \frac{1}{2}\frac{\pi^2 - 4}{\pi}$$

```
evalf(esp);
```

$$0.9341765549$$

à comparer avec la moyenne obtenue sur 1 000 tirages :

```
s:=0: to 1000 do s:=s+randF1() od: s/1000;
```

$$0.9627213171$$

### 3.3. Exercices.

1. Dans le problème considéré au §2, on suppose $p = q = 1/2$. Tracer la distribution du dernier retour à l'ordonnée 0 lorsque la partie dure 100 étapes.
2. Pour tirer rapidement une loi gaussienne à partir d'un générateur uniforme, on tire uniformément deux nombres $x$ et $y$ entre -1 et 1, jusqu'à ce que $r = x^2 + y^2 \leq 1$. On renvoie alors
$$x\sqrt{\frac{-2\log r}{r}}.$$
Comparer ce procédé à celui qui consiste à inverser
$$\frac{1}{\sqrt{2\pi}}\int_0^x e^{-t^2/2}\,dt.$$

CHAPTER XII

# Calcul formel et calcul numérique

Le calcul numérique ne s'oppose pas au calcul formel. Il en est son complément et parfois même son aboutissement. C'est le cas en mathématiques appliquées et en analyse numérique, par exemple pour le traitement du signal, l'analyse et la commande des systèmes, la robotique et la théorie de l'estimation.

Il est possible de faire du calcul numérique à l'intérieur d'un système de calcul formel ; nous en illustrons certains aspects en §1. Cependant les contraintes d'efficacité obligent souvent à utiliser d'autres langages ou logiciels. Les problèmes d'interface qui se posent alors sont abordés en §2.

## 1. Calcul numérique à l'intérieur du système

Lorsqu'une approximation numérique d'un résultat suffit, on utilise des nombres décimaux (appelés aussi nombres *flottants*). Tous les systèmes de calcul formel permettent de choisir le nombre de chiffres significatifs utilisés au cours d'un calcul. Nous parlons alors de calcul *en précision arbitraire*, par opposition aux opérations à précision fixe (usuellement 15 décimales) proposées par les autres langages (C, Fortran, Pascal) et logiciels numériques.

Certains systèmes, dont Maple, offrent également la possibilité d'accéder interactivement aux opérations flottantes de la machine. Nous montrons comment exploiter au mieux ce mode pour diminuer le temps de calcul d'un facteur pouvant aller jusqu'à 30. Si cela ne suffit pas, il faut alors passer à la production de code C ou Fortran, ce qui permet grâce à la compilation de gagner encore un facteur pouvant aller jusqu'à 100, soit un facteur d'environ 3 000 par rapport au départ. Ces questions sont abordées dans la section suivante.

**1.1. La précision arbitraire.** Lorsque le nombre de chiffres désirés est supérieur à la précision des flottants de la machine, il est nécessaire de recourir aux nombres flottants en précision arbitraire du système de calcul formel, ce qui peut s'avérer coûteux en temps. Certaines bibliothèques telles que GMP, Bignum ou Pari (voir Ann. C) font aussi ce type de calcul en grande précision, mais il faut s'attendre au mieux à un gain de l'ordre de dix. De plus GMP et Bignum savent faire uniquement les opérations arithmétiques élémentaires (addition, multiplication, soustraction, division) et n'ont pas la souplesse d'emploi d'un système interactif.

La grande précision est cependant souvent nécessaire lors de calculs instables numériquement. Dans certains cas, les calculs numériques peuvent assez

facilement être accélérés de manière non négligeable, nous l'illustrerons sur deux exemples importants.

*Phénomène d'annulation.* La précision que l'utilisateur indique au système pour les calculs flottants (en Maple par `Digits:=n` ou bien `evalf(expr,n)`) est la précision relative des calculs, c'est-à-dire que la mantisse des nombres manipulés contient $n$ chiffres.

Si $a$ et $b$ sont des approximations à $n$ chiffres de deux nombres positifs, $a+b$ est une approximation à $n$ chiffres de leur somme. La précision *relative* de $a-b$ par rapport à leur différence est quant à elle inférieure à $n$ chiffres, et ce d'autant plus que les nombres sont proches. C'est le phénomène d'annulation.

EXEMPLE 1. Dans l'exemple suivant, il se trouve que $e^{\pi\sqrt{163}}$ est égal à l'entier 262537412640768744 à $10^{-12}$ près ! Si l'on ne prend pas garde à la précision choisie, le calcul numérique de cette différence donne des résultats qui semblent incohérents lorsque le nombre de chiffres significatifs choisi est trop faible.

```
seq(evalf(exp(Pi*sqrt(163))-262537412640768744,i),
  i=[10,15,20,25,30,35]);
```

$$-.29\,10^{10}, -5000., .43, -.36\,10^{-5}, .24\,10^{-10}, -.75020\,10^{-12}$$

Le phénomène d'annulation se remarque lorsque le nombre de chiffres significatifs affichés par le système est très inférieur à la précision demandée. Ainsi dans l'exemple ci-dessus, dix chiffres sont demandés pour la première valeur $-.29\,10^{10}$, mais seulement deux sont obtenus. Cet exemple montre aussi qu'il ne faut pas faire confiance aveuglément aux chiffres affichés lorsqu'ils sont peu nombreux. Une technique efficace consiste à augmenter la précision demandée de 5 ou 10 unités et à observer la précision obtenue : si elle augmente de même, alors le résultat est sans doute fiable.

*Réglage de la précision.* Pour obtenir $n$ chiffres corrects au bout d'une suite d'opérations, il faut effectuer les calculs intermédiaires avec une précision $n' \geq n$ après avoir montré qu'elle était suffisante. Par exemple pour le calcul de constantes définies par une série, comme dans le cas du sinus intégral de FRESNEL détaillé au §VIII.1.1, $n' = n + a + b$, où $a$ prend en compte le phénomène d'annulation (fonction du rapport entre le terme maximal et la somme de la série), et $b$ prend en compte les erreurs d'arrondi (fonctions du nombre de termes ajoutés). Il suffit à la fin du calcul de tronquer le résultat obtenu à $n$ chiffres.

Mais il n'est pas toujours nécessaire de réaliser tous les calculs avec la même précision. Une illustration importante de ce principe est fournie par l'itération de NEWTON. Pour résoudre $f(x) = 0$, la méthode de NEWTON part d'un point $x_0$ puis calcule des éléments de la suite $x_{n+1} = x_n - f(x_n)/f'(x_n)$ ; sous certaines conditions, cette suite converge vers un zéro de $f$. On peut montrer que dans ce cas le nombre de décimales correctes est approximativement doublé à chaque itération.

EXEMPLE 2. Une approximation de $\sqrt{2}$ s'obtient comme racine de $f = x^2 - 2$ ce qui donne $x_{n+1} = x_n/2 + 1/x_n$. Une première méthode consiste à exécuter

```
Digits:=1000: x:=1.4: to 9 do x:=x/2+1/x od:
```

qui fournit 999 décimales correctes. Il est cependant plus habile d'effectuer

```
Digits:=1: x:=1.4:
while Digits<1000 do Digits:=2*Digits; x:=x/2+1/x od:
```

qui produit seulement 990 décimales, mais huit fois plus rapidement. En effet, dans le second cas, le coût du calcul est quasiment réduit à celui de la dernière itération.

*Utilisation d'entiers.* Il faut savoir qu'en Maple les nombres flottants sont représentés par des couples $(m, e)$ où $m$ et $e$ sont des entiers, respectivement la mantisse et l'exposant. Par conséquent tous les calculs flottants en grande précision sont basés sur les opérations entre grands entiers. Il ressort de ceci que l'on a intérêt lorsque c'est possible à utiliser directement des entiers. C'est notamment le cas lors du calcul approché d'une somme $s = \sum_i t_i$ lorsque le quotient $t_{i+1}/t_i$ est rationnel en $i$. Au lieu d'ajouter les nombres flottants $t_i$, il vaut mieux ajouter les nombres entiers $t'_i = 10^d t_i$, où $d$ est le nombre de décimales du résultat final ; $t'_{i+1}$ s'obtient à partir de $t'_i$ par multiplication et division entière.

EXEMPLE 3. La procédure ci-dessous calcule ainsi $n$ décimales de $e$, la base des logarithmes népériens :

```
Enum := proc(n) local t,s,i;
   t:=10^n; s:=t;
   for i while t<>0 do t:=iquo(t,i); s:=s+t od;
   Float(s,−n)
end:
```

Bien qu'interprétée, cette procédure est presque aussi rapide que `evalf(E,n)`, qui appelle une routine du noyau de Maple, et 35 fois plus rapide que la procédure suivante :

```
Enum2 := proc(n) local t,s,i;
   Digits:=n; t:=1.0; s:=t;
   for i while t>Float(1,-n) do t:=t/i; s:=s+t od;
   s
end:
```

En plus du gain dû à l'utilisation d'entiers au lieu de nombres flottants, le premier programme ne calcule des termes $t_i$ que la partie utile au résultat final, alors que le second les détermine tous avec $n$ chiffres significatifs.

1.1.1. *Le temps de calcul*

La principale faiblesse des systèmes de calcul formel en calcul numérique est leur lenteur. Nous l'illustrons sur un exemple de calcul des valeurs singulières d'une matrice.

EXEMPLE 4. Les valeurs singulières d'une matrice $A$ sont les racines carrées des valeurs propres de $A^tA$. Le rapport entre la plus grande valeur singulière et la plus petite, appelé le *conditionnement* de la matrice, quantifie la sensibilité de la solution $u$ du système linéaire $Au = b$ par rapport aux données $A$ et $b$. La solution est d'autant plus sensible que le conditionnement est grand. Calculons les valeurs singulières d'une matrice numérique $100 \times 100$ obtenue aléatoirement en Maple :

```
m:=evalm(randmatrix(100,100)/99.0):
time(evalf(Svd(m)));
```

33.134

Le résultat est obtenu après 33 secondes. Sauvegardons dans un fichier les éléments de la matrice utilisée et réalisons le même calcul dans un système spécialisé comme Scilab, qui exécute en fait des programmes Fortran :

```
-->m=read('mat100.data',100,100);

-->initimer(),v=svd(m);timer()
 ans       =

    0.6
```

Les résultats donnés par les deux systèmes sont les mêmes, mais le temps de calcul est de 33 secondes avec Maple, contre 0.6 seconde avec Scilab, soit un rapport de plus de 50.

La lenteur des systèmes de calcul formel pour les calculs numériques provient de deux facteurs :

(1) *généralité* : les systèmes de calcul formel utilisent des structures de données plus générales que les systèmes numériques, ce qui leur donne la possibilité de faire des calculs en précision arbitraire. Le revers de la médaille est que ces structures plus générales sont plus lourdes à manipuler, d'où un plus grand nombre d'opérations élémentaires ;

(2) *compilation* : aucun système de calcul formel actuel ne permet la compilation efficace de l'arithmétique flottante. Lorsque ces instructions sont élémentaires comme l'addition de deux nombres flottants, la compilation diminue le temps de calcul d'un facteur pouvant aller jusqu'à 100.

1.1.2. *La précision*

En revanche, dans le cas de la résolution de problèmes numériques mal conditionnés où un langage comme C ou Fortran manque de précision pour

donner la bonne solution, un système de calcul formel peut être très utile. Bien sûr, il faut que le problème ne soit pas de trop grande taille et, comme nous l'avons vu dans le paragraphe précédent, que le temps de calcul ne soit pas une contrainte impérative.

EXEMPLE 5. Reprenons le calcul des valeurs singulières d'une matrice, mais pour une matrice de HILBERT, matrice très mal conditionnée. En Maple, le calcul avec une précision de 50 chiffres est aisé :

```
h:=hilbert(30):
time(assign(v=evalf(Svd(h),50)));
                                34.784
v[1],v[15],v[16],v[17],v[30];
```

$$1.9864925686087363474534871311714227555376938069200,$$
$$.27112398180823380771764993344965366797871366951760\,10^{-14},$$
$$.98296319223573173752229056229247630363353503058785\,10^{-16},$$
$$.31914477625966150949774017342529486456377251924053\,10^{-17},$$
$$.46986365445013202024187188936355711531488366113258\,10^{-43}$$

Le résultat s'obtient en 35 secondes, et quelques valeurs singulières sont affichées ci-dessus, dont la plus grande `v[1]` et la plus petite `v[30]`.

Avec Scilab, le temps est très court (moins d'un dixième de seconde !) mais seuls `v(1)` à `v(15)` sont corrects :

```
-->for i = 1:30, for j = 1:30, h(i,j) = 1/(i+j-1);end;end
-->initimer(),v=svd(h);timer()
 ans       =
    0.

-->[v(1),v(15),v(16),v(17),v(30)]
 ans       =

! 1.9864926   2.711D-15   1.022D-16   1.465D-17   9.100D-20 !
```

En effet, les nombres flottants de Scilab, qui sont ceux de la machine, n'offrent que 16 chiffres de précision, ce qui est insuffisant ici où le rapport entre `v[1]` et `v[30]` est de l'ordre de $10^{44}$. Des résultats semblables à ceux de Scilab sont obtenus par `evalf(Svd(h),16)` sous Maple.

Les résultats obtenus par Maple ne sont pas nécessairement corrects. La vérification n'est pas simple à réaliser. Le même calcul avec 100 chiffres de précision donne les mêmes 48 premiers chiffres pour `v[1]`, 37 chiffres pour `v[15]`, 35 pour `v[16]`, 33 pour `v[17]` et 8 pour `v[30]`, mais ce n'est en aucun cas une justification rigoureuse.

**1.2. Les flottants de la machine.** Maple donne accès aux opérations sur les nombres décimaux de la machine (les quatre opérations, les fonctions trigonométriques, l'exponentielle, le logarithme,...). Leur précision est limitée

à une quinzaine de chiffres, mais ces opérations sont bien plus rapides que celles en précision arbitraire fournies par le système. Ces nombres sont souvent utilisés en Maple à l'insu de l'utilisateur. Les deux cas les plus importants sont les tracés graphiques (`plot` et `plot3d`) où cinq décimales suffisent largement, et l'intégration numérique (`evalf/int`) lorsque la précision demandée est inférieure à quinze chiffres.

EXEMPLE 6. Pour calculer la 10 000$^{\text{e}}$ itérée de la suite définie par $u_0 = 1$ et $u_{n+1} = u_n + \sin(u_n)$, une première solution est d'exécuter :

```
s:=1.0: to 10000 do s:=s+sin(s) od:
```

en affectant au préalable à la variable `Digits` le nombre de décimales voulues, plus quatre ou cinq pour se prémunir contre les erreurs d'arrondi. Si une quinzaine de décimales suffisent, la commande

```
s:=1.0: to 10000 do s:=evalhf(s+sin(s)) od:
```

est trois fois plus rapide. Mieux encore, l'évaluation d'une procédure :

```
f:=proc(u0) local u; u:=u0; to 10000 do u:=u+sin(u) od end:
s:=evalhf(f(1.0)):
```

est quatorze fois plus rapide. En effet, dans ce dernier cas la conversion entre les nombres décimaux de Maple et ceux de la machine n'est faite que deux fois, à l'entrée et à la sortie de la procédure `f`.

EXEMPLE 7. Pour tracer l'ensemble de MANDELBROT (voir p. 83), il faut calculer des valeurs de la suite $z_{n+1} = z_n^2 + z_0$, où $z_0$ est un nombre complexe donné. C'est ce que fait la procédure `f` ci-dessous, qui part de $z_0 = a_0 + ib_0$ et renvoie le carré du module de $z_n$, ou d'une valeur précédente s'il dépasse 4.

```
f := proc(a0,b0,n)
local a,b,c;
  a:=a0; b:=b0;
  to n while a^2+b^2<=4 do
    c:=a^2−b^2; b:=2*a*b+b0; a:=c+a0
  od;
  a^2+b^2
end:
```

Pour cette fonction, l'utilisation des flottants de la machine apporte un gain proche de 30 :

```
time(f(0.3,0.2,100000)), time(evalhf(f(0.3,0.2,100000)));
                              107.233,    3.683
```

Cet exemple montre que la commande `evalhf` permet d'évaluer avec des flottants de la machine des procédures relativement complexes, comprenant des boucles et des tests. Cependant nous avons atteint là les limites de ce qui est possible sans sortir de Maple. Nous verrons dans la section suivante que l'on peut encore gagner un facteur 100 grâce à la compilation (ex. 8 p. 285).

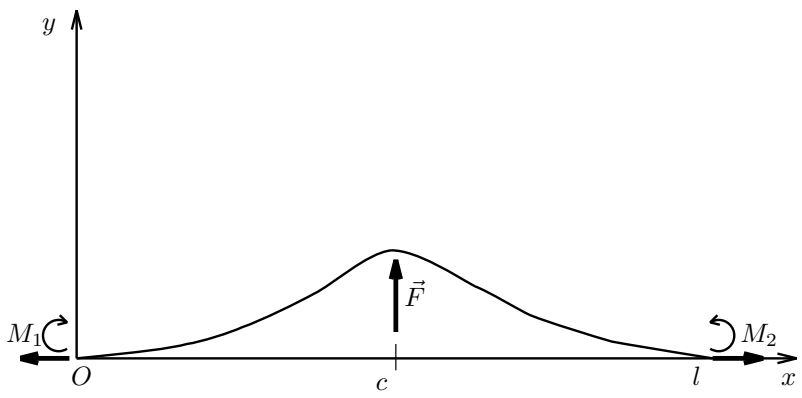



FIGURE 1 Un hauban chargé transversalement.

**1.3. Un exemple concret.** Le calcul formel est un outil idéal pour résoudre beaucoup de problèmes qui se posent aux ingénieurs. En effet, l'ingénieur a souvent des modélisations physiques compliquées et, travaillant dans des plages déterminées de variation des paramètres, est intéressé par des formules approchées, par exemple pour les insérer dans un programme. À côté de la recherche classique de telles formules par des méthodes d'identification ou par tâtonnement, le calcul formel, par l'intermédiaire des développements limités et des calculs de limites, peut donner une réponse précise et rapide.

À titre d'exemple, nous traitons ci-après un problème complet qui a trait à l'obtention d'une formule approchée pour la longueur d'un hauban. Un hauban est un câble généralement métallique qui sert de renfort. Connaissant les forces qui s'appliquent sur ce câble, un problème classique consiste à en calculer la longueur. Il est représenté dans le cas d'une charge transversale sur la figure 1.

Les variables sont les suivantes : $M_1$ le couple de flexion à l'extrémité gauche du hauban ; $M_2$ le couple de flexion à l'extrémité droite ; $T$ la force de traction aux deux extrémités ; $q$ la charge uniformément répartie le long du hauban (correspond ici au poids du hauban) ; $\vec{F}$ la charge transversale au point d'abscisse $c$ ; $l$ la longueur entre les deux extrémités ; $p$ paramètre dépendant de la nature et de la forme du matériau.

La flèche $y$ du hauban est donnée par $y(x) = y_F(x) + y_q(x) + y_M(x)$ avec :

$$y_F(x) = \begin{cases} -\frac{F\sinh(pc)\sinh(px)}{Tp\sinh(pl)} + \frac{Fcx}{Tl} & \text{si } x \le c, \\ -\frac{F\sinh[p(l-c)]\sinh[p(l-x)]}{Tp\sinh(pl)} + \frac{F(l-c)(l-x)}{Tl} & \text{si } x \ge c, \end{cases}$$

$$y_q(x) = \frac{q}{Tp^2}\left(\frac{\cosh(px - pl/2)}{\cosh(pl/2)} - 1\right) + \frac{qx(l-x)}{2T},$$

$$y_M(x) = \frac{M_1}{T}\left(\frac{l-x}{l} - \frac{\sinh[p(l-x)]}{\sinh(pl)}\right) - \frac{M_2}{T}\left(\frac{x}{l} - \frac{\sinh(px)}{\sinh(pl)}\right).$$

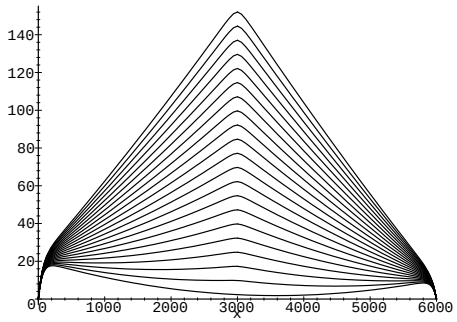



FIGURE 2 Courbes du hauban pour diverses valeurs de $F$.

Dans un premier temps, nous allons étudier la fonction $y(x)$ en fonction de la force $F$. Les autres paramètres sont donnés : $l = 6\ 000$, $c = l/2$, $p = 1{,}62 \times 10^{-2}$, $q = -0{,}283$, $T = 98\ 100$, $M_1 = 2 \cdot 10^6$ et $M_2 = -10^6$. Une première étape consiste à tracer la forme du hauban, c'est-à-dire la courbe $y(x)$ pour diverses valeurs de $F$, par exemple pour $F$ variant de 0 à 10 000 par pas de 500. La session Maple ci-dessous permet de tracer les 21 courbes reproduites dans la figure 2.

```
y[F]:=-F*sinh(p*c)*sinh(p*x)/(T*p*sinh(p*l))+F*c*x/(T*l):
y[F]:=[y[F],subs([x=l-x,c=l-c],y[F])]:
y[q]:=q*(cosh(p*x-p*l/2)/cosh(p*l/2)-1)/T/p/p+q*x*(l-x)/(2*T):
y[M]:=(M[1]/T)*((l-x)/l-sinh(p*(l-x))/sinh(p*l))
     -(M[2]/T)*(x/l-sinh(p*x)/sinh(p*l)):
y:=[y[F][1]+y[q]+y[M],y[F][2]+y[q]+y[M]]:
data:=c=l/2,l=6E3,T=98100,p=1.62E-2,q=-.283,M[1]=2E6,M[2]=-1E6:
v:=subs(data,y):
p1:=plot(map(subs,{seq(F=500*i,i=0..20)},v[1]),x=0..3000):
p2:=plot(map(subs,{seq(F=500*i,i=0..20)},v[2]),x=3000..6000):
plots[display]({p1,p2});
```

Le but final est de calculer la longueur du hauban, qui est donnée par la formule :

$$L = \int_0^c \sqrt{1 + {y_1'}^2}\, dx + \int_c^l \sqrt{1 + {y_2'}^2}\, dx$$

où $y_1'$ et $y_2'$ représentent respectivement les dérivées de $y_1$ (flèche à gauche de $x = c$) et $y_2$ (flèche à droite de $x = c$) par rapport à $x$.

Maple ne trouve pas d'expression symbolique pour cette intégrale. Nous allons donc la simplifier en observant que pour les valeurs numériques données ${y'}^2$ est petit devant 1. En effet, la figure 3 montre que pour $F = 10\ 000$, le maximum de ${y'}^2$ vaut environ 0,14.

```
dy:=[diff(y[1],x)^2,diff(y[2],x)^2]:
v:=subs(data,F=10000,dy):
plots[display]({plot(v[1],x=0..3000),plot(v[2],x=3000..6000)});
```

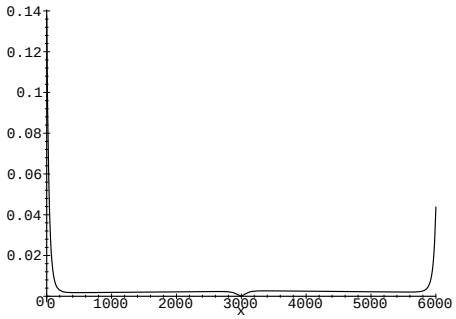



FIGURE 3 Courbe donnant le carré de la dérivée de la flèche.

En remplaçant $\sqrt{1+y'^2}$ par $1+y'^2/2$ (la légitimité de cette approximation sera vérifiée *a posteriori*), la longueur $L$ devient la valeur approchée :

$$L_{ap} = l + \frac{1}{2}\left(\int_0^c {y_1'}^2\,dx + \int_c^l {y_2'}^2\,dx\right)$$

expression qui se calcule formellement.

```
Lap:=l+(int(dy[1],x=0..c)+int(dy[2],x=c..l))/2:
length(Lap);
```

157288

L'expression obtenue est très grosse et difficilement maniable pour être utilisée dans un programme de calcul numérique. Parmi les sous-expressions qui reviennent fréquemment (que l'on observe dans les variables annexes %) se trouve $e^{-lp}$, qui vaut $0{,}6 \times 10^{-42}$ avec les valeurs numériques indiquées plus haut. Ceci suggère de simplifier `Lap` en utilisant le fait que $e^{-lp}$ est très petit.

Pour pouvoir effectuer la simplification $e^{-lp} \to 0$, le plus simple est de commencer par exprimer les fonctions trigonométriques hyperboliques en termes d'exponentielles (tab. 5 p. 25), puis de développer l'expression obtenue pour faire apparaître tous les produits $lp$ suffisamment isolés pour que la substitution $lp \to \ln(t)$ soit possible par un `subs` (voir chap. II). Pour avoir la formule approchée, il suffit alors de calculer la limite de l'expression obtenue lorsque $t$ tend vers l'infini. Au passage nous simplifions encore un peu l'expression en tenant compte du fait que $l = c/2$.

```
Lap:=limit(subs(c=l/2,l*p=ln(t),expand(convert(Lap,exp))),
       t=infinity);
```

$$\begin{aligned} Lap := \big(&-24p^3M_1M_2 - 12p^3{M_1}^2 + 3p^3F^2l^2 + p^3q^2l^4 - 48pFql \\ &+ 12M_1ql^2p^3 - 24q^2l^2p + 3p^3Fl^3q - 12M_2Flp^3 + 24T^2p^3l^2 \\ &- 12p^3{M_2}^2 + 60q^2l - 9p^2F^2l + 6p^4{M_2}^2l - 12M_2ql^2p^3 + 36M_2qp^2l \\ &-36M_1qp^2l + 6p^4M_1^2l + 12M_1Flp^3\big)\,/(24lT^2p^3) \end{aligned}$$

L'expression obtenue contient 19 termes. La validité des approximations est confirmée par une comparaison à l'intégration numérique directe de l'expression de la longueur exacte :

```
evalf(subs(data,F=10000,[Lap,
  Int(sqrt(1+dy[1]),x=0..c)+Int(sqrt(1+dy[2]),x=c..l)]));
```

$$[6009.704162, 6009.653189]$$

Si l'on a un grand nombre de longueurs de poutre à calculer le temps gagné dans la simple évaluation de l'expression ci-dessus par rapport au calcul numérique de l'intégrale est considérable (en Maple le rapport est de 60). Pour s'assurer de la validité des approximations à chaque jeu de données, il faudrait bien sûr calculer aussi une formule approchée de l'erreur commise.

La section suivante revient plus en détail sur la manière de produire le code Fortran correspondant à la longueur approchée `Lap` :

```
fortran(subs(M[1]=M1,M[2]=M2,[L=Lap]));
```

```
      L = (3*p**3*F**2*l**2+36*M2*q*l*p**2-12*p**3*M2**2+12*p**3*F*l*M1-
     #12*p**3*M1**2-12*p**3*F*l*M2+24*p**3*T**2*l**2+3*p**3*F*l**3*q+6*p
     #**4*M1**2*l-9*p**2*F**2*l-24*p**3*M1*M2-12*p**3*q*M2*l**2+6*p**4*M
     #1**2*l+p**3*q**2*l**4-36*q*M1*p**2*l+12*p**3*q*M1*l**2-48*q*F*l*p+
     #60*q**2*l-24*q**2*l**2*p)/l/p**3/T**2/24
```

## 2. Lien avec d'autres langages ou bibliothèques

Pour certains calculs numériques, la lenteur des systèmes de calcul formel est rédhibitoire, et le recours à des langages tels que C ou Fortran est indispensable. C'est le cas notamment des applications en temps réel, comme la commande d'une fusée, où un délai de quelques secondes dans la résolution d'un système peut s'avérer fatal.

En revanche, lors de la préparation du programme embarqué, un système de calcul formel constitue un environnement très agréable pour la mise en équations, ne faisant pas d'erreur dans les calculs de dérivées, et fournissant des optimisations non négligeables.

Depuis un système de calcul formel, il est donc important de savoir produire du code numérique. Nous distinguons deux modes de production de code :

(1) l'évaluation d'une expression : l'utilisateur insère ces expressions dans un programme qu'il écrit lui-même ;
(2) la production de programmes : le système de calcul formel produit un programme complet, prêt à être compilé, à partir d'une description sous forme de macro-instructions fournie par l'utilisateur.

Avant d'illustrer ces deux modes de production de code, nous montrons l'utilisation depuis un système de calcul formel de petits programmes numériques que l'utilisateur a lui-même écrits.

**2.1. Utilisation de sous-programmes en C ou Fortran.** Lorsqu'une petite boucle numérique utilise la majeure partie du temps total de calcul, il peut s'avérer fructueux d'écrire cette boucle dans un langage compilé tel que C ou Fortran et de lier le sous-programme ainsi réalisé au système de calcul formel.

La technique la plus efficace de transmission de données entre le système de calcul formel et le langage numérique consiste à utiliser la même représentation interne des données. Ceci n'est possible que si le système de calcul formel est assez ouvert. Dans sa version de base, Maple ne permet pas une telle communication, et nous n'en parlerons pas plus ici.

La technique la plus employée est de communiquer sous un format intermédiaire que le système de calcul formel et le langage numérique savent lire et écrire (le plus souvent par l'intermédiaire d'un fichier texte). À cause de ces recopies, cette technique n'est intéressante que lorsque le nombre d'opérations est élevé par rapport au volume des données à transmettre (en entrée ou en sortie du sous-programme).

Cette section dépend assez fortement du système d'exploitation utilisé. Les indications données ci-dessous s'appliquent au système Unix.

Exemple 8. Le calcul de termes d'une suite numérique est un exemple idéal. Considérons la fonction `f` de l'exemple 7 p. 280. Un programme C réalisant le même calcul est le suivant :

```
#include <stdio.h>
#include <stdlib.h>

main(int argc,char *argv[])
{
   double a,b,c,a0=atof(argv[1]),b0=atof(argv[2]);
   int n=atoi(argv[3]);

   for (a=a0, b=b0; n && a*a+b*b<=4; n--) {
         c=a*a-b*b; b=2*a*b+b0; a=c+a0;
   }
   printf(§"%e\n",a*a+b*b);
}
```

Les arguments $a_0$, $b_0$ et $n$ sont donnés sur la ligne de commande. Une fois le programme compilé, son exécution est très simple :

```
% mandelbrot 0.3 0.2 100000
1.767315e-01
```

Depuis Maple cet exécutable s'utilise comme un sous-programme ainsi :

```
readlib(C):
ff := proc(a0,b0,n)
   system(cat(‘mandelbrot ‘,‘C/float‘(a0),‘ ‘,
              ‘C/float‘(b0),‘ ‘,n,‘ > resultat‘));
```

```
   op(readdata(resultat))
end:

ff(0.3,0.2,100000);
```

.176731500000000014

Le temps de calcul du sous-programme en C est environ 37 millisecondes et le temps de transfert des données est de 2 millisecondes, soit un temps total voisin de 0,04 seconde, contre 107 s en Maple et 3,7 s en Maple avec les flottants de la machine. La compilation nous a donc permis d'accélérer d'un facteur 100 notre programme.

**2.2. Code évaluant une expression.** Avant d'illustrer les deux modes de production de code (expressions et programmes), nous décrivons un exemple typique de problème numérique : la résolution d'un système non linéaire.

*Résolution d'un système non linéaire.* Un système non linéaire à $n$ équations et $n$ inconnues se présente sous la forme

$$\begin{cases} f_1(x_1,\ldots,x_n) & = 0 \\ \vdots & \vdots \\ f_n(x_1,\ldots,x_n) & = 0 \end{cases}$$

avec $x_1,\ldots,x_n$ appartenant à $\mathbb{R}$. Ce système s'écrit aussi sous forme matricielle $F(X)=0$ après avoir posé

$$X=\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \qquad \text{et} \qquad F(X)=\begin{pmatrix} f_1(x_1,\ldots,x_n) \\ \vdots \\ f_n(x_1,\ldots,x_n) \end{pmatrix}.$$

Une façon de résoudre un tel système est d'utiliser la méthode de Newton généralisée, extension de la méthode de Newton pour la recherche de zéros d'une fonction. Cet algorithme peut s'écrire, toujours en notation matricielle :

$X := X_0$
**while** $norm(F(X)) > \epsilon$ **do**
  $<$ trouver $Y$ solution du système linéaire $F'(X).Y = -F(X) >$
  $X := Y + X$
**end**

où $F'(X)$ représente la matrice jacobienne du système, c'est-à-dire la matrice dont l'élément $(i,j)$ est $\partial f_i/\partial x_j$. Il faut donc calculer $n(n+1)/2$ dérivées partielles (la matrice est symétrique) avant de pouvoir écrire le programme de résolution, ou d'utiliser un programme tout fait qui prend en entrée la matrice jacobienne.

Si le problème est assez compliqué, ce simple calcul de dérivées partielles peut s'avérer fastidieux et surtout amener des erreurs. La solution qui consiste

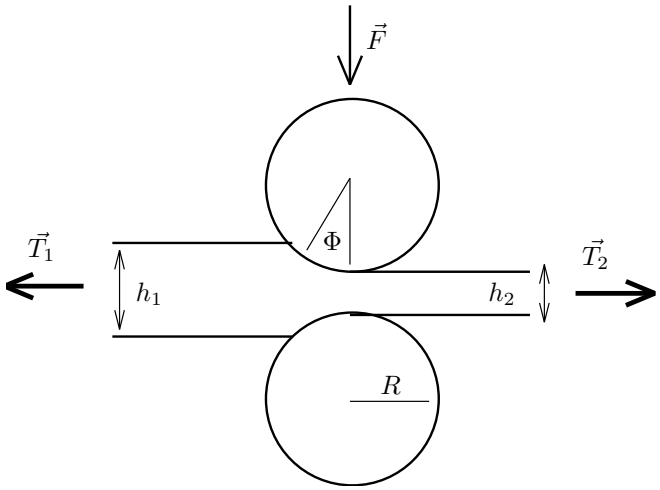



FIGURE 4 Une cage de laminage.

à calculer ces valeurs par discrétisation, par exemple en approchant

$$\frac{\partial f_i(x_1,\ldots,x_n)}{\partial x_j} \quad \text{par} \quad \frac{f_i(x_1,\ldots,x_j+h_j,\ldots,x_n)-f_i(x_1,\ldots,x_j,\ldots,x_n)}{h_j}$$

n'est pas non plus très satisfaisante car elle amène des erreurs qui rendent la convergence de la méthode de NEWTON plus lente. La solution idéale consiste à utiliser un système de calcul formel pour calculer ces dérivées, et à lui faire imprimer directement les dérivées partielles dans le langage choisi pour éviter les erreurs de recopie.

*Code calculant une expression.* Nous illustrons la production de code calculant une expression sur la résolution d'un système non linéaire par un exemple précis emprunté au domaine de l'ingénierie, à savoir la modélisation d'une cage d'un train de laminage à chaud. Le même exemple illustrera aussi la production de programmes.

Un train de laminage à chaud a pour but de réduire l'épaisseur d'une tôle d'acier chaude en la faisant passer entre des cylindres. Le dispositif où se trouvent les cylindres se nomme une cage et en général la tôle passe à travers un certain nombre de cages successives (fig. 4).

Les équations qui régissent le comportement de la cage sont les suivantes :

$$
\begin{aligned}
f_1(F,h_2,\Phi) &= h_2 - S - \frac{F + a_2(1-e^{a_3 F})}{a_1} \\
f_2(F,h_2,\Phi) &= F + \frac{R\xi T_1}{h_2} \\
&\quad - lkR\left(\frac{1}{2}\pi\sqrt{\frac{h_2}{R}}\arctan(\sqrt{r}) - \frac{\pi\xi}{4} - \ln\left(\frac{h_N}{h_2}\right) + \frac{1}{2}\ln\left(\frac{h_1}{h_2}\right)\right) \\
f_3(F,h_2,\Phi) &= \arctan\left(\Phi\sqrt{\frac{R}{h_2}}\right) \\
&\quad - \frac{1}{2}\sqrt{\frac{h_2}{R}}\left(\frac{\pi}{4}\ln\left(\frac{h_2}{h_1}\right) + \sqrt{\frac{R}{h_2}}\arctan(\sqrt{r}) - \frac{T_1}{klh_1} + \frac{T_2}{klh_2}\right) \\
\text{avec} \qquad & r = \frac{h_1 - h_2}{h_2}, \qquad \xi = \sqrt{\frac{h_1 - h_2}{R}}, \qquad h_N = h_2 + R\Phi^2.
\end{aligned}
$$

Les inconnues sont la force de laminage $F$, l'épaisseur de sortie $h_2$ et l'angle neutre $\Phi$ (déterminant le point où la vitesse de glissement de la tôle par rapport au cylindre est nulle). Les autres variables sont données, en particulier l'épaisseur d'entrée $h_1$, les tractions $T_1$ et $T_2$, le module de cédage $a_1$, le serrage des vis $S$, la résistance $k$ du métal à la déformation et la largeur d'entrée $l$ de la tôle. Les coefficients $a_2$ et $a_3$ ont été trouvés expérimentalement par les lamineurs.

Il est à noter que ces équations sont simplifiées car elles ne tiennent pas compte de la température de la tôle. Pour le problème réel, il faut en plus résoudre une équation de la chaleur le long de la tôle.

Il est clair que le calcul de la matrice jacobienne du système est pour le moins compliqué. En revanche il est très facile à réaliser en utilisant un système de calcul formel comme Maple. À titre de curiosité, l'élément $(2,2)$ de la matrice jacobienne est imprimé ci-dessous.

```
xi:=sqrt((h1-h2)/R):
r:=(h1-h2)/h2:
hN:=h2+R*Phi^2:
f:=[h2-S-(F+a2*(1-exp(a3*F)))/a1,
    F-l*k*R*(Pi*sqrt(h2/R)*arctan(sqrt(r))/2-Pi*xi/4
      -log(hN/h2)+log(h1/h2)/2)+R*xi*T1/h2,
    arctan(Phi*sqrt(R/h2))-sqrt(h2/R)*(Pi*log(h2/h1)/4
      +sqrt(R/h2)*arctan(sqrt(r))-T1/k/l/h1+T2/k/l/h2)/2]:
Fp:=linalg[jacobian](f,[F,h2,Phi]):
Fp[2,2];
```

$$
\begin{aligned}
-lkR\Bigg(&\frac{\pi \arctan\left(\sqrt{\frac{h_1-h_2}{h_2}}\right)}{4\sqrt{\frac{h_2}{R}}R}+\frac{\pi\sqrt{\frac{h_2}{R}}\left(-\frac{1}{h_2}-\frac{h_1-h_2}{{h_2}^2}\right)}{4\sqrt{\frac{h_1-h_2}{h_2}}\left(1+\frac{h_1-h_2}{h_2}\right)}\\
&+\frac{\pi}{8\sqrt{\frac{h_1-h_2}{R}}R}-\frac{\left(\frac{1}{h_2}-\frac{h_2+R\Phi^2}{{h_2}^2}\right)h_2}{h_2+R\Phi^2}-\frac{1}{2h_2}\Bigg)\\
&-\frac{T_1}{2\sqrt{\frac{h_1-h_2}{R}}h_2}-\frac{R\sqrt{\frac{h_1-h_2}{R}}T_1}{{h_2}^2}
\end{aligned}
$$

Le problème peut être beaucoup plus compliqué qu'une simple modélisation du laminoir, par exemple si l'on s'intéresse à sa régulation. En réalité, il y a généralement sept cages successives, donc la modélisation demande la résolution des équations précédentes sept fois de suite ; de plus le temps intervient et la simulation doit être faite à chaque pas de temps. La régulation demande donc une simulation rapide du système, voire en temps réel, et celle-ci ne peut être réalisée que dans un langage numériquement rapide comme Fortran ou C.

Transformons donc la matrice jacobienne obtenue précédemment en syntaxe Fortran. En Maple, la commande `fortran` réalise cette transformation. Si son argument est un tableau, tous les éléments sont transformés.

```
fortran(Fp);
```

```
      Fp(1,1) = -(1-a2*a3*exp(a3*F))/a1
      Fp(1,2) = 1
      Fp(1,3) = 0
      Fp(2,1) = 1
      Fp(2,2) = -l*k*R*(0.3141593E1/sqrt(h2/R)*atan(sqrt((h1-h2)/h2))/R/
     #4+0.3141593E1*sqrt(h2/R)/sqrt((h1-h2)/h2)*(-1/h2-(h1-h2)/h2**2)/(1
     #+(h1-h2)/h2)/4+0.3141593E1/sqrt((h1-h2)/R)/R/8-(1/h2-(h2+R*Phi**2)
     #/h2**2)/(h2+R*Phi**2)*h2-1/h2/2)-1/sqrt((h1-h2)/R)*T1/h2/2-R*sqrt(
     #(h1-h2)/R)*T1/h2**2
      Fp(2,3) = 2*l*k*R**2*Phi/(h2+R*Phi**2)
      Fp(3,1) = 0
      Fp(3,2) = -Phi/sqrt(R/h2)*R/h2**2/(1+Phi**2*R/h2)/2-1/sqrt(h2/R)*(
     #0.3141593E1*alog(h2/h1)/4+sqrt(R/h2)*atan(sqrt((h1-h2)/h2))-T1/k/l
     #/h1+T2/k/l/h2)/R/4-sqrt(h2/R)*(0.3141593E1/h2/4-1/sqrt(R/h2)*atan(
     #sqrt((h1-h2)/h2))*R/h2**2/2+sqrt(R/h2)/sqrt((h1-h2)/h2)*(-1/h2-(h1
     #-h2)/h2**2)/(1+(h1-h2)/h2)/2-T2/k/l/h2**2)/2
      Fp(3,3) = sqrt(R/h2)/(1+Phi**2*R/h2)
```

Il ne faut pas oublier qu'un langage tel que Fortran ne fait pas la différence entre lettres majuscules et minuscules : contrairement à Maple, les noms `R` et `r` désignent la même variable en Fortran.

Maple propose également des possibilités d'optimisation pour éviter de recalculer les sous-expressions communes aux éléments de la matrice, comme

$\sqrt{h_2/R}$ ou $h_2 + R\Phi^2$. Maple introduit alors un certain nombre de variables auxiliaires qui contiennent ces expressions communes.

```
fortran(Fp,optimized);
```

```
      t8 = l*k
      t9 = 1/R
      t11 = sqrt(h2*t9)
      t12 = 1/t11
      t14 = h1-h2
      t15 = 1/h2
      t16 = t14*t15
      t17 = sqrt(t16)
      t18 = atan(t17)
      t22 = 1/t17
      t23 = h2**2
      t24 = 1/t23
      t26 = -t15-t14*t24
      t29 = 1/(1+t16)
      t33 = sqrt(t14*t9)
      t34 = 1/t33
      t37 = Phi**2
      t38 = R*t37
      t39 = h2+t38
      t42 = 1/t39
      t54 = R**2
      t59 = sqrt(R*t15)
      t60 = 1/t59
      t62 = R*t24
      t65 = 1/(1+t38*t15)
      t68 = 1/h1
      t73 = 1/k
      t75 = 1/l
      t78 = T2*t73
      Fp(1,1) = -(1-a2*a3*exp(a3*F))/a1
      Fp(1,2) = 1
      Fp(1,3) = 0
      Fp(2,1) = 1
      Fp(2,2) = -t8*R*(0.3141593E1*t12*t18*t9/4+0.3141593E1*t11*t22*t26*
     #t29/4+0.3141593E1*t34*t9/8-(t15-t39*t24)*t42*h2-t15/2)-t34*T1*t15/
     #2-R*t33*T1*t24
      Fp(2,3) = 2*t8*t54*Phi*t42
      Fp(3,1) = 0
      Fp(3,2) = -Phi*t60*t62*t65/2-t12*(0.3141593E1*alog(h2*t68)/4+t59*t
     #18-T1*t73*t75*t68+t78*t75*t15)*t9/4-t11*(0.3141593E1*t15/4-t60*t18
     #*t62/2+t59*t22*t26*t29/2-t78*t75*t24)/2
      Fp(3,3) = t59*t65
```

Malgré le nombre important de variables auxiliaires introduites, cette optimisation est loin d'être inutile, puisque le programme est accéléré d'un facteur variant entre 2 et 5 selon les machines.

Pour le langage C, il faut faire respectivement `C(Fp)` et `C(Fp,optimized)`, après avoir chargé le programme de transformation par `readlib(C)`.

**2.3. Production de programmes.** Le défaut de cette utilisation du calcul formel dans l'écriture d'un programme est qu'il faut aller et venir entre l'éditeur de texte et le système de calcul formel, car ce dernier génère uniquement les calculs d'expressions ; tout le reste du programme (initialisation, boucles, appels de sous-programmes) doit être écrit par l'utilisateur, ce qui peut entraîner des erreurs.

Il est possible de rester à l'intérieur du système de calcul formel pour produire tout le programme. Une première méthode consiste à utiliser les commandes d'impression du système. En Maple par exemple, une boucle `do` de Fortran est produite par :

```
lprint(‘      do 1,i=1,n‘),fortran([z=z+f(i)^2]),
lprint(‘1     continue‘);
```

ce qui produit le texte ci-dessous.

```
      do 1,i=1,n
      z = z+f(i)**2
1     continue
```

Cette façon de faire est plutôt fastidieuse. La plupart des systèmes de calcul formel procurent de quoi automatiser plus ou moins cette tâche. Maple dispose dans la *share library* de deux modules, Macrofort et MacroC, qui génèrent respectivement de véritables programmes Fortran et C.

Le principe est de décrire le programme sous la forme d'une liste dont chaque élément est la description d'une instruction ou d'un ensemble d'instructions Fortran ou C. Cette description est elle aussi sous la forme d'une liste dont le premier élément est un mot-clé décrivant l'instruction.

Par exemple, pour obtenir un sous-programme Fortran qui calcule la matrice jacobienne de la cage du laminoir avec comme arguments les variables $F$, $h_2$, $\Phi$, $T_1$ et $T_2$, et un programme principal qui appelle ce sous-programme[†], Macrofort s'emploie ainsi :

```
with(share): readshare(macrofor,numerics); init_genfor():
```

MACROFORT FORTRAN code generator Version 1.2.4

```
data:={a1=610,a2=648,a3=-.00247,l=1250,k=1.4E-2,R=360,
       h1=24,S=12}:
vFp:=subs(data,op(Fp)):
prog:=[programm,cage,
        [[declarem,doubleprecision,[F,h2,Phi,T1,T2,j(3,3)]],
         [equalf,F,1363], [equalf,h2,15], [equalf,Phi,0.06],
         [equalf,T1,12], [equalf,T2,35],
         [dom,i,1,100000,
           [callf,jacob,[F,h2,Phi,T1,T2,j]]]]]:
```

[†]Ici le programme principal effectue une simple boucle avec les mêmes paramètres. Un tel programme permet d'évaluer le gain dû à l'option `optimized` de la commande `fortran`.

```
ssprog:=[subroutinem,jacob,[F,h2,Phi,T1,T2,j],
          [[declarem,doubleprecision,[F,h2,Phi,T1,T2,j(3,3)]],
             [matrixm,j,vFp]]]:
optimized:=true: precision:=double:
genfor(prog), genfor(ssprog);
```

et l'on obtiendra

```
c
c     MAIN PROGRAM cage
c
      program cage
        doubleprecision F,h2,Phi,T1,T2,j(3,3)
        F = 1363
        h2 = 15
        Phi = 0.6D-1
        T1 = 12
        T2 = 35
c
        do 1000, i=1,100000
          call jacob(F,h2,Phi,T1,T2,j)
 1000   continue
c
      end
c
c     SUBROUTINE jacob
c
      subroutine jacob(F,h2,Phi,T1,T2,j)
        doubleprecision F,h2,Phi,T1,T2,j(3,3)
      t3 = sqrt(360.D0)
      t4 = 0.3141592653589793D1*t3
      t5 = sqrt(h2)
      t6 = 1/t5
      t7 = 24-h2
      t8 = 1/h2
      t9 = t7*t8
      t10 = sqrt(t9)
      t11 = atan(t10)
      t16 = h2**2
      t17 = 1/t16
      t23 = 1/t10*(-t8-t7*t17)/(1+t9)
      t26 = sqrt(1.D0/15.D0-h2/360)
      t27 = 1/t26
      t29 = Phi**2
      t30 = h2+360*t29
      t33 = 1/t30
      t43 = sqrt(t8)
      t44 = 1/t43
      t48 = 1/(1+360*t29*t8)
      t54 = t3*t43
```

```
      j(1,1) = -1.D0/610.D0-0.2623868852D-2*exp(-0.247D-2*F)
      j(1,2) = 1
      j(1,3) = 0
      j(2,1) = 1
      j(2,2) = -0.4375D1*t4*t6*t11-0.4375D1*t4*t5*t23-0.21875D1*0.314
    +1592653589793D1*t27+0.63D4*(t8-t30*t17)*t33*h2+0.315D4*t8-t27*T1*t
    +8/2-360*t26*T1*t17
      j(2,3) = 0.4536D7*Phi*t33
      j(3,1) = 0
      j(3,2) = -Phi*t3*t44*t17*t48/2-t3*t6*(0.3141592653589793D1*log(
    +h2/24)/4+t54*t11-0.2380952381D-2*T1+0.5714285714D-1*T2*t8)/1440-t3
    +*t5*(0.3141592653589793D1*t8/4-t3*t44*t11*t17/2+t54*t23/2-0.571428
    +5714D-1*T2*t17)/720
      j(3,3) = t54*t48
     end
```

Pour faire la même chose en langage C, les instructions à exécuter sont les suivantes.

```
with(share): readshare(macroC,numerics); init_genC():
data:={a1=610,a2=648,a3=-.00247,l=1250,k=1.4E-2,R=360.,
       h1=24,S=12}:
vFp:=subs(data,op(Fp)):
prog:=[[includeC,‘<math.h>‘],[functionm,‘‘,main,[],
         [[declareC,double,[F,h2,Phi,T1,T2,j[4,4]]],
          [declareC,int,[i]],
          [equalC,F,1363], [equalC,h2,15.0], [equalC,Phi,0.06],
          [equalC,T1,12], [equalC,T2,35],
          [form,&=(i,0),i<100000,&=(i,i+1),
           [callC,jacob,[F,h2,Phi,T1,T2,j]]
      ]]]]:
ssprog:=[functionm,‘‘,jacob,[[double,[F,h2,Phi,T1,T2,j[4,4]]]],
             [[matrixm,j,op(vFp)]]]:
optimized:=true: precision:=double: autodeclare:=double:
genC(prog), genC(ssprog);
```

Les seules différences notables sont l'inclusion du fichier `math.h` pour la déclaration des fonctions mathématiques `log`, `exp`, `atan`, `sqrt`, et l'ajout de l'instruction `autodeclare:=double` pour que les variables auxiliaires introduites par Maple soient de type `double`.

L'avantage de cette approche est non seulement d'éviter la manipulation fastidieuse de fichiers, mais aussi de permettre d'écrire des codes génériques dont les entrées sont des expressions symboliques. On peut par exemple écrire une fonction Maple qui, à partir d'un système non linéaire de $n$ équations à $n$ inconnues, produit automatiquement le code Fortran ou C réalisant la résolution de ce système selon la méthode de Newton généralisée.

Une autre façon de résoudre ce problème de lien entre numérique et formel est d'appeler à partir du système de calcul formel des bibliothèques numériques existantes. C'est le cas par exemple du système IRENA, qui

fonctionne à l'intérieur du système de calcul formel Reduce, et qui permet d'appeler les sous-programmes Fortran de la bibliothèque NAG sans sortir de Reduce.

**2.4. Lien avec les bibliothèques numériques.** Un système de CAO en automatique, en anglais *CACSD package (Computer Aided Control System Design)*, est un logiciel qui gère une grande bibliothèque de programmes Fortran et C pour le calcul numérique matriciel, l'intégration des systèmes d'équations différentielles, la commande et l'optimisation des systèmes et le traitement du signal. Il dispose d'un interpréteur permettant d'écrire des programmes – appelés *macros* – et tracer des courbes. Le plus connu de ces systèmes est le logiciel Matlab. Citons aussi le logiciel français Basile et le logiciel Scilab réalisé à l'INRIA (voir p. 307).

Lorsqu'on lie un système de calcul formel avec un système de CAO en automatique, le premier réalise les calculs symboliques et le second est adapté aux calculs numériques performants à l'aide des programmes Fortran et C qu'il gère. Ce lien peut être fait de façon plus ou moins automatique. Il existe par exemple un lien entre Maple et Matlab. C'est une boîte à outils (ensemble de fonctions) rajoutée à Matlab qui permet, tout en restant dans Matlab, d'appeler Maple, de réaliser des calculs et de récupérer les résultats en Matlab. Un autre type de lien consiste à produire du code Fortran à partir du système de calcul formel et à l'introduire dans le système de CAO comme nouvelle fonction.

Il est difficile de présenter un exemple en détail car les calculs mécaniques sont souvent longs et nécessitent des connaissances qui sortent du cadre de ce livre. Nous allons simplement donner les grandes lignes d'une application récente d'un tel lien entre Maple et le système de CAO Scilab. Ce dernier système fonctionne sur stations de travail Unix avec X-Windows. Il est librement distribué, sources compris. De plus, il est possible d'ajouter facilement des programmes C ou Fortran à Scilab.

L'application présentée consiste à réaliser la simulation et la commande d'un vélo. Les commandes sont les couples appliqués au guidon et à la roue arrière. Le problème de simulation consiste, la position de départ du vélo et les valeurs des commandes étant données, à chercher la trajectoire suivie. Pour le problème de commande, la position de départ est donnée et il s'agit de déterminer les commandes qui permettent d'atteindre un état donné le plus rapidement possible, par exemple de tourner l'axe du vélo d'un angle fixé.

La technique utilisée consiste à employer le formalisme de Lagrange. Ceci nécessite le calcul d'une matrice $23 \times 23$, de plusieurs gradients, et la résolution d'un immense système différentiel linéaire. Les calculs de matrices et de gradients sont réalisés en Maple ; des programmes Fortran sont ensuite produits de façon automatique en utilisant Macrofort (voir §2.3), puis ils sont compilés et inclus dans Scilab sous la forme de macros. Ils peuvent alors être efficacement utilisés par les fonctions de Scilab. À titre de curiosité, nous

donnons ci-après la valeur d'un élément de matrice du problème de commande, montrant ainsi la quasi-impossibilité d'une écriture à la main

```
    fmat(41,67) = -(pd(18)*t257+2*pd(18)*t252+t804+t810+t814+t819+t
+192-2*pd(18)*t240+t677-pd(18)*t229+pd(18)*t199+t166+t655+t669+t598
++t617+t625+t637+t608+2*pd(18)*t181+t580+t588+t593+t568+t573+t555+t
+550+t531+t516-pd(18)*t1125+t507+pd(18)*t1137+t483+t493+t475+t467-2
+*pd(18)*t1148+t454+2*pd(18)*t1154+t437+t540-pd(18)*t786+pd(18)*t11
+94+pd(18)*t755+pd(18)*t773-pd(18)*t721+pd(18)*t717-pd(18)*t901-2*p
+d(18)*t896-pd(18)*t889+2*pd(18)*t1210+pd(18)*t855-pd(18)*t874+t408
++t398+t394+t388+t379+t361+t369+t337+pd(18)*t10*param(17)*lambda(7)
+*t148+2*pd(14)*param(20)*lambda(14)*t3*t171-pd(18)*t10*param(19)*l
+ambda(9)*t148+pd(18)*t10*param(17)*lambda(7)*t219+t301+2*pd(18)*t1
+0*param(17)*lambda(7)*t206-2*pd(18)*t10*param(19)*lambda(9)*t206-p
+d(18)*t10*param(19)*lambda(9)*t219-pd(18)*t10*param(19)*lambda(10)
+*t244+pd(14)*param(19)*lambda(9)*t525+pd(14)*param(20)*lambda(15)*
+t9*t684+pd(14)*param(19)*lambda(9)*t725+pd(14)*param(17)*lambda(7)
+*t641-2*pd(14)*param(17)*lambda(7)*t737-pd(14)*param(19)*lambda(9)
+*t462-pd(14)*param(19)*lambda(9)*t641-pd(14)*param(17)*lambda(7)*t
+525+t28*pd(19)*t132*param(19)*lambda(10)*t122+2*t133*pd(19)*t132*p
+aram(19)*lambda(10)*t828+2*t133*pd(19)*t10*param(1)*t915+t28*pd(19
+)*t132*param(19)*lambda(9)*t309+2*t133*pd(19)*t132*param(19)*lambd
+a(9)*t846+t28*pd(19)*t10*param(1)*t933+2*pd(14)*param(19)*lambda(9
+)*t737+t134*pd(19)*t132*param(19)*lambda(10)*t908+t28*pd(19)*t10*p
+aram(1)*t866-t28*pd(19)*t132*param(17)*lambda(7)*t309-2*t133*pd(19
+)*t132*param(17)*lambda(7)*t846+t134*pd(19)*t10*param(20)*lambda(1
+5)*t821-2*t133*pd(19)*t132*param(17)*lambda(6)*t828+t28*pd(19)*t10
+*param(20)*lambda(15)*t123-pd(14)*param(19)*lambda(10)*t517+pd(14)
+*param(20)*lambda(14)*t3*t684-pd(18)*t132*param(20)*lambda(15)*t12
+2-param(1)*pd(20)*t10*t766-2*pd(14)*param(19)*lambda(9)*t760+t134*
+pd(19)*t132*param(19)*lambda(9)*t840+2*t133*pd(19)*t10*param(20)*l
+ambda(14)*t950-t134*pd(19)*t132*param(17)*lambda(7)*t840-t28*pd(19
+)*t132*param(17)*lambda(6)*t122-2*pd(14)*param(19)*lambda(10)*t102
+3-2*pd(14)*param(19)*lambda(10)*t1033+pd(14)*param(17)*lambda(7)*t
+462-pd(14)*param(20)*lambda(14)*t1017+pd(14)*param(20)*lambda(14)*
+t3*t1072+pd(14)*param(20)*lambda(15)*t1107-pd(14)*param(19)*lambda
+(10)*t1165-pd(14)*param(17)*lambda(7)*t725-pd(18)*t132*param(20)*l
+ambda(14)*t309+param(1)*pd(20)*t10*t9*t702-param(1)*pd(20)*t10*t3*
+t689+2*pd(18)*t10*param(17)*lambda(6)*t1185+pd(18)*t10*param(17)*l
+ambda(6)*t123+pd(14)*param(17)*lambda(6)*t1165+pd(14)*param(17)*la
+mbda(6)*t517-2*pd(18)*t132*param(20)*lambda(14)*t3*t204+pd(18)*t10
+*param(17)*lambda(6)*t244-pd(18)*t10*param(19)*lambda(10)*t123-2*p
+d(18)*t10*param(19)*lambda(10)*t1185-2*pd(18)*t132*param(20)*lambd
+a(15)*t9*t204-pd(18)*t132*param(20)*lambda(14)*t618-pd(18)*t132*pa
+ram(20)*lambda(15)*t225+2*pd(14)*param(17)*lambda(6)*t1033+pd(14)*
+param(20)*lambda(15)*t640+2*pd(14)*param(20)*lambda(15)*t3*t1054+p
```

Le résultat de la simulation ou de la commande est une animation graphique du comportement du vélo. La figure 5 p. 296 donne la trajectoire optimale

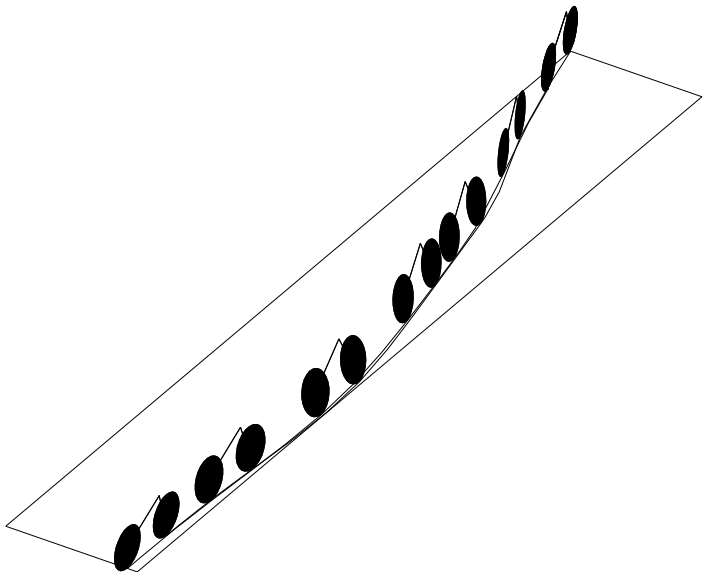


FIGURE 5 Comment faire tourner un vélo de 15 degrés le plus rapidement possible.

d'un vélo pour tourner de 15 degrés (noter le contrebraquage au départ, qui permet de tourner plus vite).

L'intérêt de cette approche est sa flexibilité : on peut ainsi résoudre un grand nombre de problèmes variés avec la méthode que l'on désire sans être lié à un système spécialisé. De plus les systèmes utilisés, Maple et Scilab, sont très efficaces dans leur domaine respectif. L'ensemble fournit un environnement riche pour des problèmes de commande de systèmes.

2.4.1. *Les logiciels spécialisés*

Pour conclure ce chapitre, mentionnons l'existence d'un certain nombre de logiciels spécialisés qui réalisent des liens entre calcul formel et calcul numérique dans des domaines bien particuliers. En mécanique, ces logiciels permettent en général d'écrire les équations, de les transformer en utilisant un formalisme approprié, puis de les résoudre en produisant habituellement du code Fortran ou C.

Certains de ces logiciels sont indépendants de tout système de calcul formel et incluent donc les opérations nécessaires à leur domaine d'application,

d'autres sont construits autour d'un système de calcul formel existant. Pour l'analyse des systèmes multicorps ou polyarticulés, dans le premier cas, on peut citer des logiciels tels que Mesa-Verde, Dynamica ou Autolev. Dans le second cas on peut citer les logiciels James et Sygmmae qui sont construits au-dessus de Maple.

Plus de détails sur ces systèmes sont présentés en annexe C.

# Annexes

ANNEXE A

# Une session Mathematica

Cette annexe reprend l'exemple du chapitre I en Mathematica. Elle illustre l'analogie entre les commandes mises à la disposition de l'utilisateur par les différents systèmes de calcul formel.

Comme en Maple, la première étape consiste à définir les équations des deux cercles

```
In[1]:= eq1 = x^2+y^2==R1^2;
In[2]:= eq2 = (x-R1)^2+y^2 == R2^2;
```

(ici le symbole = désigne l'affectation, le symbole == l'égalité et le point-virgule ordonne à Mathematica de ne pas afficher le résultat). Le système est ensuite résolu par rapport à $x$ et $y$ :

```
In[3]:= sol = Solve[{eq1,eq2},{x,y}]
```

$$\text{Out}[3] = \{\{y \to \frac{-R2\sqrt{4 - R2^2/R1^2}}{2}, x \to \frac{2R1^2 - R2^2}{2R1}\},$$
$$\{y \to \frac{R2\sqrt{4 - R2^2/R1^2}}{2}, x \to \frac{2R1^2 - R2^2}{2R1}\}\}$$

Les solutions sont présentées comme un ensemble de substitutions. La solution cherchée (celle pour laquelle $y$ est positif) est la seconde, c'est-à-dire `sol[[2]]` en Mathematica. On en déduit les valeurs des sinus et cosinus de $\theta_1$ et $\theta_2$.

```
In[4]:= sint1 = Together[y/Sqrt[x^2+y^2] /. sol[[2]]]
```

$$\text{Out}[4] = \frac{R2\sqrt{\frac{4R1^2 - R2^2}{R1^2}}}{2\sqrt{R1^2}}$$

```
In[5]:= sint2 = Together[y/Sqrt[(R1-x)^2+y^2] /. sol[[2]]]
```

$$\text{Out}[5] = \frac{R2\sqrt{\frac{4R1^2 - R2^2}{R1^2}}}{2\sqrt{R2^2}}$$

```
In[6]:= cost1 = Together[x/Sqrt[x^2+y^2] /. sol[[2]]]
```

$$\text{Out}[6] = \frac{2R1^2 - R2^2}{2R1\sqrt{R1^2}}$$

```
In[7]:= cost2 = Together[(R1-x)/Sqrt[(R1-x)^2+y^2] /. sol[[2]]]
```

$$\text{Out}[7] = \frac{R2^2}{2R1\sqrt{R2^2}}$$

L'expression `expr /. sol[[2]]` signifie `expr` dans lequel on a effectué la substitution `sol[[2]]`, c'est-à-dire remplacé $x$ et $y$ en fonction de $R_1$ et $R_2$. La commande `Together` effectue la réduction au même dénominateur.

```
In[8]:= A=PowerExpand[R1^2(ArcCos[cost1]-sint1 cost1)+
                      R2^2(ArcCos[cost2]-sint2 cost2)]
```

$$\text{Out}[8] = R2^2\left(\frac{-R2\sqrt{4R1^2-R2^2}}{4R1^2}+\text{ArcCos}\left(\frac{R2}{2R1}\right)\right)+ R1^2\left(\frac{-R2(2R1^2-R2^2)\sqrt{4R1^2-R2^2}}{4R1^4}+\text{ArcCos}\left(\frac{2R1^2-R2^2}{2R1^2}\right)\right)$$

La commande `PowerExpand` développe des expressions contenant des puissances. Le symbole de multiplication est facultatif en Mathematica. Voici comment on vérifie les valeurs extrêmes de l'aire :

```
In[9]:= A /. R2->0
```

$$\text{Out}[9] = 0$$

```
In[10]:= A /. R2->2 R1
```

$$\text{Out}[10] = \pi R1^2$$

Le rapport $P$ des aires est défini par :

```
In[11]:= AA = Expand[A /. R2 -> K R1]
```

$$\text{Out}[11] = \frac{-KR1\sqrt{4R1^2-K^2R1^2}}{2}+K^2R1^2\,\text{ArcCos}(K/2) + R1^2\,\text{ArcCos}\left(\frac{2R1^2-K^2R1^2}{2R1^2}\right)$$

```
In[12]:= P = Together[PowerExpand[Simplify[AA / (Pi R1^2)]]]
```

$$\text{Out}[12] = \frac{-K\sqrt{4-K^2}+2K^2\,\text{ArcCos}(K/2)+2\,\text{ArcCos}(1-K^2/2)}{2\pi}$$

La recherche de la valeur de $K$ donnant $P = 1/2$ se fait à l'aide de la commande `FindRoot` de résolution numérique des équations transcendantes

```
In[13]:= FindRoot[P==1/2,{K,1},WorkingPrecision->17]
```

$$\text{Out}[13] = \{K \to 1.15872847301812152\}$$

à laquelle on a donné $K = 1$ comme point de départ et demandé 17 chiffres de précision.

ANNEXE B

# Autour du calcul formel

### Informations électroniques

En France, le calcul formel est un domaine où la recherche est très vivante. Les principaux domaines d'activité concernent les polynômes, les équations différentielles, les applications en combinatoire et les interfaces. Deux groupements coordonnent une partie de ces recherches, ce sont le GDR (groupe de recherche) Médicis et le pôle calcul formel du PRC (programme de recherches coordonnées) mathématiques et informatique.

Une partie de ces activités a pour retombée un serveur ftp

`medicis.polytechnique.fr (129.104.3.60)`

qui contient de nombreuses informations utiles sur le calcul formel. La même machine propose également un serveur WEB spécialisé en calcul formel à partir duquel il est possible d'accéder à d'autres serveurs dans le monde (URL `http://medicis.polytechnique.fr/`).

Les personnes ayant accès aux *news* peuvent y lire un *newsgroup* entièrement consacré au calcul formel : `sci.math.symbolic`. Par ailleurs, les personnes ayant accès au courrier électronique peuvent demander par simple *email* (voir la documentation en ligne de `maple_group`) à faire partie de la *Maple User Group mailing list*.

La *share library* de Maple contient des programmes développés par des utilisateurs. Elle est distribuée avec le système, mais les nouvelles versions de la *share library* sont plus fréquentes que les nouvelles versions de Maple. On peut trouver la dernière version de la *share library* par ftp sur la machine

`ftp.inria.fr (192.93.2.54)`

dans le répertoire `lang/maple`.

### Revues et conférences

La principale revue du domaine s'appelle *The Journal of Symbolic Computation*. L'association américaine ACM édite également un bulletin, le *SIGSAM Bulletin*, consacré au calcul formel. La *Maple Technical Newsletter*, publiée par Birkhäuser, est une revue consacrée uniquement à Maple, qui contient des articles de niveau élémentaire sur des applications de Maple, en particulier pour l'enseignement.

Enfin, les conférences ayant trait au calcul formel sont :

- ISSAC *(International Symposium on Symbolic and Algebraic Computation)*, conférence annuelle dont les actes sont généralement publiés par ACM Press ;
- AAECC *(Applied Algebra and Error Correcting Codes)*, conférence bisannuelle plus orientée vers l'algèbre, dont les actes sont publiés dans la série *Lecture Notes in Computer Science*, Springer-Verlag ;
- DISCO *(Design and Implementation of Symbolic Computation Systems)*, conférence annuelle dont les actes sont publiés dans la série *Lecture Notes in Computer Science*, Springer-Verlag.

ANNEXE C

# Index des systèmes actuels

Les systèmes que nous décrivons ici sont les plus couramment utilisés. Il est bien clair qu'il en existe beaucoup d'autres et que l'on ne saurait être exhaustif.

Pour chaque système nous indiquons son type (commercial ou gratuit), la version actuelle, les machines sur lesquelles il est disponible (stations de travail Unix, Macintosh ou PC), le distributeur principal, et les principales caractéristiques.

## 1. Systèmes généraux

**Axiom**. Type : commercial. Version actuelle : 2. Disponible sur Sun SPARC Station, IBM RS/6000, HP 9000, Silicon Graphics (une version pour PC est prévue pour fin 1995). Distributeur : NAG Ltd, Wilkinson House, Jordan Hill Road, Oxford, OX2 8DR, Angleterre, fax (0)865 310139, tél. (0)865 511245, email `infodesk@nag.co.uk`. Système comprenant le mieux les concepts mathématiques, du fait de son caractère fortement typé ; inclut l'implantation la plus complète de l'algorithme d'intégration de Risch ; possibilité de faire des calculs paresseux. Mise en œuvre lourde pour un non-spécialiste.

**Derive**. Type : commercial. Version actuelle 2.59. Disponible sur PC et certaines calculatrices de poche (HP 95). Distributeur : Soft Warehouse, 3660 Waialae Avenue, Suite 304, Honolulu, HI 96816-3236, USA, tél. (808) 734-5801, fax (808) 735-1105. Petit logiciel peu gourmand en mémoire pouvant même fonctionner sur certains ordinateurs de poche. Langage de programmation rudimentaire, pas d'accès aux structures de données, factorisation des polynômes faible.

**Macsyma**. Type : commercial. Version actuelle 419.0 sous Unix et 2.0 sur PC. Disponible sur machines Unix et PC (PC Macsyma). Distributeur : Macsyma Inc., 20 Academy Street, Suite 201, Arlington, MA 02174-6436, USA, tél. (617) 646-4550, fax (617) 646-3161, email `info@macsyma.com`. Il existe de nombreux systèmes dérivés : Maxima, Aljabr, Vaxima, Paramacs. Le système qui compte le plus d'hommes-années de développement, doté de nombreux modules spécialisés qui n'existent pas encore dans les autres systèmes. Système un peu vieillissant.

**Maple**. Type : commercial. Version actuelle : V.3. Disponible sur machines Unix, Mac et PC. Distributeur : Waterloo Maple Software, 450 Phillip Street, Waterloo, Ontario, N2L 5J2, Canada, tél. (519) 747-2373, fax (519) 747-5284, email `info@maplesoft.on.ca`. Facilité de mise en œuvre, rapidité et accès aux sources des bibliothèques.

**Mathematica**. Type : commercial. Version actuelle 2.2. Disponible sur machines Unix, Mac et PC. Distributeur : Wolfram Research Inc., 100 Trade Center Drive, Champaign, IL 61820, USA, email `info@wri.com`, fax 217-398-0747. Possibilité d'utiliser des sous-programmes en C ou Fortran via le logiciel MathLink. Programmation par règles de récriture, mais peu efficace. Un peu trop boîte noire.

**MuPAD**. Type : gratuit (pour les établissements à caractère non commercial). Version actuelle 1.2.1. Disponible sur machines Unix, Mac et PC sous Linux (`ftp.uni-paderborn.de`). Distributeur : MuPAD-Distribution, Fachbereich 17, Univ. of Paderborn, Warburger Str. 100, D-33095 Paderborn, Allemagne, email `MuPAD-Distribution@uni-paderborn.de`. Permet la programmation orientée objets, comprend des instructions de parallélisme explicite. Système jeune, certaines fonctionnalités manquent.

**Reduce**. Type : commercial. Version actuelle 3.5. Disponible sur machines Unix, Mac et PC. Distributeur : The Rand Corporation, 1700 Main Street, P.O. Box 2138, Santa Monica, CA 90407-2138, USA, tél. (310) 393-0411 poste 7681, fax (310) 393-4818, email `reduce@rand.org`. L'un des plus vieux systèmes, donc très robuste. Reduce a été en grande partie développé par ses utilisateurs, et manque d'organisation globale.

## 2. Systèmes spécialisés

**Gap**. Calculs sur les groupes finis. Type : gratuit. Version actuelle 3.3 (ftp `ftp.math.rwth-aachen.de`). Disponible sur machines Unix, Mac et PC. Distributeur : Lehrstuhl D fuer Mathematik, RWTH Aachen, 52056 Aachen, Allemagne, email `gap@math.rwth-aachen.de`.

**GB**. Calculs de bases de Gröbner (bases standard). Type : gratuit (ftp `posso.ibp.fr`). Version actuelle : 4.0. Disponible sur machines Unix. Écrit en C++ par Jean-Charles Faugère, Université de Paris VI, `jcf@posso.ibp.fr`. Très rapide. Utilise notamment de nouveaux algorithmes qui ne sont implantés dans aucun autre système, et est conçu pour pouvoir être utilisé comme serveur de calcul de bases de Gröbner.

**Macaulay**. Calculs en géométrie algébrique (anneaux de polynômes sur des corps finis). Type : gratuit (ftp `math.harvard.edu`). Version actuelle : 3.0. Disponible sur machines Unix, Mac et PC. Auteurs : Dave Bayer, Department of Mathematics, Barnard College, New York, NY 10027, USA, tél. (212) 854-2643, `dab@math.columbia.edu`, et Mike Stillman, Department of Mathematics, Cornell University, Ithaca, NY 14853, USA, tél. (607) 255-7240, email `mike@math.cornell.edu`.

**Magma**. Calculs sur les ensembles, groupes, anneaux, corps, algèbre linéaire et combinatoire. Type : commercial. Version actuelle : V1. Disponible sur machines Unix (une version pour PC 486/Pentium est annoncée pour fin 1994). Distributeur : The Secretary, Computational Algebra Group, School of Mathematics and Statistics, University of Sydney, NSW 2006, Australie, fax 2-351-4534, tél. 2-351-3338, email `magma@maths.su.oz.au`.

**Pari/GP**. Calculs en théorie des nombres. Type : gratuit. Version actuelle : 1.38 (ftp `megrez.math.u-bordeaux.fr`). Disponible sur machines Unix, Mac et PC. Écrit en C par Christian Batut, Dominique Bernardi, Henri Cohen, Michel Olivier (Univ. de Bordeaux), `pari@ceremab.u-bordeaux.fr`. Dispose de nombreuses fonctionnalités générales comme la recherche numérique des racines de polynômes, les tracés graphiques, les manipulations de matrices, et d'un mini-langage de programmation. Est utilisable aussi comme une bibliothèque de fonctions arithmétiques en C (les systèmes de calcul formel Axiom et MuPAD utilisent Pari pour leur arithmétique). Très efficace.

## 3. Systèmes de CAO en automatique

Ce ne sont pas des systèmes de calcul formel. Il ne font en effet que des calculs numériques, même si certains manipulent des polynômes et des matrices de fractions rationnelles (à coefficients numériques). Cependant, leur mode d'utilisation est très proche de celui des systèmes de calcul formel et ils réalisent très efficacement la partie numérique (principalement le calcul matriciel, la résolution de systèmes différentiels et l'optimisation) et le tracé de courbes.

Ceux décrits ci-après ont la possibilité d'échanger des données avec Maple.

**Basile**. Type : commercial. Version actuelle : 4.1. Disponible sur machines Unix, VMS et PC. Distributeur : Simulog, 1 rue James Joule, 78286 Guyancourt Cedex, France, tél. 30 12 27 00, fax 30 12 27 27. Réalisé en collaboration avec l'INRIA. Ce système, aux fonctionnalités très complètes, a l'avantage d'être complètement français avec un fort support local.

**Matlab**. Type : commercial. Version actuelle : 4.1. Disponible sur machines Unix, Mac et PC. Distributeur : The MathWorks, Inc., 24 Prime Park Way, Natick, MA 01760-1500, USA, tél. (508) 653-1415, fax (508) 653-2997, email `info@mathworks.com`. Le plus diffusé actuellement.

**Scilab**. Type : gratuit (ftp `ftp.inria.fr`). Version actuelle : 2.0. Disponible sur machines Unix. Développé à l'INRIA-Rocquencourt par le projet Meta2 en collaboration avec le laboratoire Cergrene de l'ENPC, email `scilab@inria.fr`. Il possède la plupart des fonctionnalités que l'on attend d'un tel type de système, en particulier il permet la manipulation symbolique des systèmes linéaires.

# Bibliographie

## Ouvrages généraux

Une bonne introduction aux mathématiques du calcul formel :
Davenport (James H.), Siret (Yves), et Tournier (Évelyne). – *Calcul formel*. – Masson, Paris, 1993, seconde édition.

La seule référence actuelle sur les principaux algorithmes du calcul formel :
Geddes (Keith O.), Czapor (Stephen R.), et Labahn (George). – *Algorithms for Computer Algebra*. – Kluwer Academic Publishers, 1992.

Sur Maple, les livres de référence sont les trois ouvrages des développeurs publiés chez Springer-Verlag : le plus gros, *Maple V: Library Reference Manual*, est entièrement contenu dans la documentation en ligne. Un ouvrage introductif sur Maple est :
Heck (André). – *Introduction to Maple*. – Springer-Verlag, 1993.

À l'heure actuelle il n'existe pas d'ouvrage sur la programmation en Maple.

## Ouvrages plus spécialisés

Un ouvrage très complet mais un peu ardu qui traite plus en détail et qui va plus loin sur de nombreux aspects de ce livre, notamment sur la première partie du chapitre IV et du chapitre VI :
Cohen (Henri). – *A course in computational algebraic number theory*. – Springer-Verlag, 1993, *Graduate Texts in Mathematics*, vol. 138.

Un livre plus élémentaire sur une partie de ces sujets est :
Zippel (Richard). – *Effective polynomial computation*. – Kluwer Academic Publishers, 1993.

Un petit livre extrêmement agréable à lire, mais malheureusement épuisé, pour aller plus loin en combinatoire :
Comtet (Louis). – *Analyse Combinatoire*. – Presses Universitaires de France, Paris, 1970. 2 volumes.

Il est parfois possible d'en trouver l'édition anglaise (également épuisée) :
Comtet (Louis). – *Advanced Combinatorics*. – Reidel, Dordrecht, 1974.

Les fonctions génératrices sont présentes dans de nombreux chapitres de ce livre, une introduction pédagogique y est donnée dans
Wilf (Herbert S.). – *Generatingfunctionology*. – Academic Press, 1990.

Sur les polynômes en calcul formel et leur utilisation en géométrie, l'ouvrage suivant est extrêmement clair et complet :
Cox (David), Little (John), et O'Shea (Donal). – *Ideals, Varieties, and Algorithms*. – Springer-Verlag, 1992.

Pour toutes les questions d'analyse mentionnées dans ce livre, une bonne référence est
Dieudonné (Jean). – *Calcul Infinitésimal*. – Hermann, Paris, 1968.

Un livre très intéressant et agréable de calcul asymptotique par l'exemple :
De Bruijn (Nicolaas G.). – *Asymptotic Methods in Analysis*. – Dover, 1981. Une réimpression de la troisième édition par North Holland, en 1970 (première édition, 1958).

Un livre de recettes pour résoudre des équations différentielles :
Zwillinger (Daniel). – *Handbook of Differential Equations*. – Academic Press, 1989.

Sur les probabilités, voici deux classiques, le premier est élémentaire mais a peut-être un peu vieilli :
Rényi (Alfred). – *Calcul des probabilités*. – Dunod, 1966.

Le second est plus complet mais plus gros :
Feller (William). – *An Introduction to Probability Theory and its Applications*. – John Wiley, 1968, troisième édition, vol. 1.

# Index