Calcul formel

par **Claude GOMEZ**
Ancien Élève de l'École Centrale de Paris
Docteur Ingénieur
Directeur de Recherche à l'Institut National de Recherche en Informatique et Automatique (INRIA)

Le calcul formel est de plus en plus connu dans le monde des scientifiques et en particulier dans celui des ingénieurs. Cela est dû en partie à la « démocratisation » de son utilisation. En effet, il y a quelques années, seule une grosse configuration d'ordinateur permettait de faire fonctionner correctement les systèmes de calcul formel existants. De nos jours, ces systèmes fonctionnent raisonnablement sur des micro-ordinateurs à faible coût (PC, Macintosh). Ensuite, sous l'impulsion du système de calcul formel Mathematica, une grande publicité a été faite pour ces systèmes, les faisant ainsi connaître du grand public scientifique. Aujourd'hui, presque tout utilisateur d'un ordinateur peut se procurer, à un prix raisonnable, un système de calcul formel.

Lorsque l'on vient d'acquérir un tel système, il est très facile, dans un premier temps, de réaliser des calculs simples, du style « calculatrice formelle », mais ensuite on veut généralement aller plus loin et, là, une certaine connaissance du système et de ses limitations est indispensable pour éviter le découragement de l'utilisateur. Du temps de formation est donc nécessaire pour une utilisation optimale d'un système de calcul formel.

Alors, une autre question apparaît : « le calcul formel est-il utile pour moi ? » ; autrement dit, « est-il rentable pour moi de passer du temps à apprendre à utiliser un tel système ? ». Le but de ce chapitre est de répondre à cette question. Pour cela, nous allons passer en revue les principaux domaines des mathématiques dans lesquels le calcul formel peut résoudre des problèmes. Ces domaines sont ceux où l'ingénieur a généralement à travailler : les calculs sur les nombres et les fractions rationnelles, la dérivation, la simplification de formules et les tracés de courbes qui sont la base de tout système de calcul formel, les calculs intégral et matriciel, la résolution d'équations couramment utilisées par les ingénieurs et, enfin, le calcul numérique. Ce dernier est en général la fin du travail de l'ingénieur et le calcul formel s'avère considérablement utile dans ce domaine ; nous insistons particulièrement sur ce point. Pour chaque partie, nous montrons ce que sait faire le calcul formel, comment il le fait et quelles sont ses limitations.

Un grand nombre d'exemples émaillent le chapitre, afin de montrer le fonctionnement du calcul formel à travers un système. Nous avons choisi le ***système de calcul formel Maple*** *(version V.3) pour cela, car c'est un système très largement diffusé (avec* ***Mathematica****), qu'il dispose d'une bibliothèque suffisamment riche et ouverte (le programme source de la plupart des fonctions est accessible) et qu'il est aisément extensible.*

Le but de ce chapitre n'est pas la description du système de calcul formel Maple. Nous n'expliquerons pas de façon détaillée la syntaxe et le fonctionnement de ce système. Mais les exemples ont été choisis pour qu'ils soient compréhensibles par le lecteur ; des explications sont données chaque fois que cela est nécessaire.

Fonctionnement d'un système de calcul formel comme Maple*. L'utilisateur entre une commande, terminée par un point virgule « ; » dans une syntaxe très naturelle, et* ***Maple*** *affiche la réponse en format* ***haute résolution*** *qui ressemble à la typographie mathématique. Si l'on remplace le point virgule par deux points « : », la réponse n'est pas affichée. Par ailleurs,* ***Maple*** *utilise le principe des* ***packages****, c'est-à-dire qu'un grand nombre de commandes sont classées par groupes de même fonctionnalité. Dans ce cas, l'appel de la commande s'écrit <****nom du package****> [<****nom de la commande****>], comme* linalg[det].

1. Calculs de base

Les fonctionnalités de base du calcul formel sont celles que l'on retrouve, en général, dans tout système de calcul formel. En particulier, ce sont celles que l'on utilise lorsque l'on se sert d'un système de calcul formel comme d'une calculette : c'est le mode que l'on peut appeler *calculatrice formelle*. Un simple calcul à faire, qu'il soit formel ou numérique, comme le calcul d'une dérivée ou une simplification de formule à réaliser, et le réflexe *calcul formel* assure rapidité et sûreté. De plus, cet outil permet le tracé de courbes et de surfaces d'une façon aussi conviviale que possible.

1.1 Nombres

1.1.1 Nombres entiers et rationnels

La caractéristique première du calcul formel est de manipuler des **nombres entiers et rationnels exacts,** à la différence des langages de programmation habituels (C ou Fortran).

Un calcul typique de nombres entiers est le calcul de la factorielle.

Exemple : calculons 100 ! en *Maple*. La fonction ifactor permet la décomposition d'un entier en facteurs premiers, et « " » fait référence au résultat calculé précédemment.

```
100!;
  933262154439441526816992388562667004907 1\
      596826438162146859296389521759999321\
      299156089414639761565182862536979201\
      827223758251185210916864000000000001\
      0000000000000
ifactor (");
  (2)^97 (3)^48 (5)^24 (7)^16 (11)^9 (13)^7  (17)^5 (19)^5 (23)^4
      (29)^3 (31)^3 (37)^2 (41)^2 (43)^2 (47)^2 (53) (59)
      (61) (67) (71) (73) (79) (83) (89) (97)
```

Les calculs de nombres de combinaisons et d'arrangements sont aussi faciles à réaliser en utilisant des nombres rationnels.

Par **exemple**, la probabilité de gagner le gros lot au loto est l'inverse du nombre de tirages possibles, sachant qu'un tirage consiste à tirer 6 boules parmi 49 :

```
1/binomial (49,6);
```

$$\frac{1}{13983816}$$

1.1.2 Nombres algébriques et transcendants

Le calcul formel utilise des **constantes mathématiques vraies,** qui ne sont pas des représentations flottantes, comme le nombre π ou la base des logarithmes népériens e.

Par **exemple**, utiliser l'expression Pi en *Maple* signifie bien que l'on utilise le nombre transcendant correspondant et non pas une approximation :

```
sin(Pi/5);
```

$$\frac{1}{4}\sqrt{2}\sqrt{5-\sqrt{5}}$$

1.1.3 Nombres complexes

Le calcul formel utilise les nombres complexes et permet de réaliser des calculs sur ces derniers. Pour cela, le nombre i est en général représenté par une variable. C'est I en *Maple* où le nombre complexe $a + ib$ s'écrit a + I*b.

Des fonctions permettent de passer de la représentation $\rho e^{i\theta}$ d'un nombre complexe à la représentation $a + ib$ et réciproquement.

Exemple : en *Maple*, on peut calculer la racine carrée de i :

```
sqrt(I);
```

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

et ensuite avoir sa représentation polaire :

```
polar (");
```

$$\text{polar}\quad\left(1,\frac{1}{4}\,\pi\right)$$

Le résultat de ce calcul pose le problème des racines des nombres complexes. $\sqrt{i}$ a deux racines alors que le calcul précédent n'en donne qu'une. Ce problème intervient souvent en calcul formel et nous en parlons plus amplement plus loin (§ 1.4.2).

De plus, dans un calcul avec des expressions complexes, un système de calcul formel suppose généralement que les variables qui interviennent sont réelles. Il faut donc être très prudent avec la manipulation des nombres complexes.

1.1.4 Nombres flottants

Les calculs se font avec un **nombre de chiffres déterminé** défini par l'utilisateur, ce qui permet entre autres des calculs avec une grande précision que l'on ne peut pas faire directement avec un langage du style C ou Fortran.

Exemple : on peut calculer la valeur numérique de :

$$e^{\pi\sqrt{163}} - 262537412640768744$$

avec successivement 10 (le défaut), puis 50 chiffres significatifs. Pour cela, nous utilisons en *Maple* la commande evalf avec le nombre de chiffres en deuxième argument :

```
evalf(exp(Pi*sqrt(163))–262537412640768744,10);
```

$$-.29\,10^{10}$$

```
evalf(exp(Pi*sqrt(163))–262537412640768744,50);
```

$$-.74992740280181431135\,10^{-12}$$

Exemple : Comme le nombre transcendant $e^{\pi\sqrt{163}}$ est égal au nombre entier 262537412640768744 à 10^{-12} près, il faut une grande précision pour obtenir un résultat correct, ce qui est impossible à faire en standard avec un langage numérique.

1.2 Polynômes et fractions rationnelles

Les calculs sur les polynômes et les fractions rationnelles à une ou plusieurs variables sont les calculs de base du calcul formel.

Il faut noter tout d'abord qu'un grand nombre de calculs sur des expressions qui ne sont pas des fractions rationnelles sont en fait, des calculs sur les fractions rationnelles.

Exemple : le calcul suivant :

$$\frac{1}{\sin a+\tan b}+\frac{1}{\sin a-\tan b}=\frac{2\sin a}{\sin^2 a-\tan^2 b}$$

n'est autre que le calcul rationnel :

$$\frac{1}{x+y}+\frac{1}{x-y}=\frac{2x}{x^2-y^2}$$

On peut distinguer **trois catégories d'opérations** que le calcul formel peut effectuer sur les polynômes et les fractions rationnelles.

- D'abord, nous avons **les opérations purement syntaxiques,** c'est-à-dire qui n'effectuent aucun calcul et parcourent simplement l'expression pour en extraire l'information cherchée.

Pour réaliser ces opérations correctement, il faut généralement que les polynômes soient développés par rapport à la variable d'intérêt.

Exemple : en *Maple*, le premier des calculs de degré suivants donne un résultat faux et le second donne le bon, alors que les polynômes sont les mêmes :

```
degree((x–1)^2*x–x^3+x);
```

$$3$$

```
degree (–2*x^2+2*x);
```

$$2$$

- Ensuite, nous avons les **opérations de réécriture** qui peuvent ordonner et développer les polynômes. Cela est très important et permet, en général, de donner à une expression la forme que l'on désire. La plupart des systèmes de calcul formel disposent pour réaliser ces opérations de réécriture d'un grand nombre de fonctions que l'utilisateur doit connaître.

Exemple : si l'on veut ordonner une fraction rationnelle en fonction d'une variable et que ses coefficients soient simplifiés, on fait en *Maple* :

```
f:=(1+x*(1+y))^2/(1–y^2);
```

$$f := \frac{(1+x(1+y))^2}{1-y^2}$$

```
collect (f,x,normal);
```

$$-\frac{(1+y)x^2}{y-1}-2\,\frac{x}{y-1}-\frac{1}{-1+y^2}$$

Pour ces deux premiers types d'opérations, syntaxiques et de réécriture, la limitation n'est due qu'à la taille des données que l'on manipule.

- Enfin, nous avons les **opérations qui nécessitent un calcul**.

Là des algorithmes très sophistiqués interviennent. En effet, les méthodes classiques de calcul de pgcd par exemple, à l'aide de l'algorithme d'Euclide, sont bien trop inefficaces et gourmandes en place mémoire.

Les opérations classiques que le calcul formel sait réaliser sur les polynômes sont la division euclidienne, le calcul de pgcd, le calcul de résultants, la factorisation et la résolution d'équations polynomiales. Nous verrons cette dernière (§ 4.1).

Il est à noter que tous ces calculs se font généralement dans le corps des rationnels $\mathbb{Q}$, ou dans des extensions algébriques de ce corps, en grande partie pour des raisons théoriques.

Exemple : le calcul suivant de factorisation ne peut avoir lieu en *Maple* qu'après conversion des nombres flottants en rationnels :

```
factor(x^4–0.05*x^2+0.3*x^3+0.525*x–3.15);
Error, (in factor/factor) floats not handled
factor(convert(x^4–0.05*x^2+0.3*x^3+0.525*x–3.15,
      rational));
```

$$\frac{1}{40}(2x+3)(5x-6)(4x^2+7)$$

La **factorisation** peut servir à simplifier des expressions ou à trouver des zéros de polynômes, mais il faut éviter si possible de l'utiliser car elle est très coûteuse en temps.

Le **calcul de résultants** permet, étant donné deux polynômes P et Q, de trouver un polynôme R qui s'annule si et seulement si les deux polynômes ont une racine commune (sans qu'il y ait nécessairement réciprocité). Ce calcul peut avoir des applications pratiques.

Exemple : on cherche les points d'intersection des deux courbes définies par :

$$(x^2+y^2)^3-4x^2y^2=0 \quad \text{et} \quad y^2(1+x)-(1-x)^3=0$$

qui sont tracées sur la figure **1**.

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);
```

$(-4x^7-60x^6-1+95x^3+9x-35x^2-164x^4+152x^5)^2$,
$16y^{14}+6\,032y^{12}-1\,624y^{10}+4\,192y^8-815y^6-301y^4$
$-9y^2+1$

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. Aux 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'intersection 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 traiterons les problèmes de la recherche de zéros de polynômes et de la résolution de systèmes de polynômes plus loin (§ 4.1). Notons toutefois qu'un outil puissant pour ce dernier problème est l'utilisation des bases de Gröbner ou bases standards.

1.3 Dérivation

Le **calcul de dérivées** est à la portée de tout un chacun. En effet, la méthode utilisée est facile à mette en œuvre par une application des règles connues. Ce n'est pas pour cela que ce calcul est simple.

Le calcul formel permet d'effectuer le calcul de dérivées très rapidement et surtout avec la certitude du résultat juste.

Exemple : en *Maple* :

```
diff(x^(x^x),x);
```

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

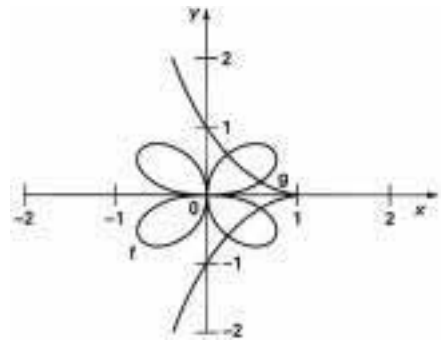

Figure 1 – Intersections de deux courbes

Le **calcul de dérivées partielles** n'est pas plus difficile.

Mais, on peut aussi utiliser le calcul formel pour faire des calculs compliqués de **dérivées formelles**.

Dans le cas suivant, on veut calculer les dérivées par rapport à x, y et z de $a(x,y,z)$, qui interviennent par la suite dans des calculs compliqués :

$$a(x,y,z)=\int_0^y \frac{\partial}{\partial x} w(x,t)\,dt+\int_{k(x)}^{g(x)} u(x,t)\,dt.$$

Mais on veut pouvoir changer les valeurs de $u(x,y)$, $w(x,y)$, $k(x)$ et $g(x)$. Il vaut donc mieux faire les calculs formellement, puis ensuite donner des valeurs à ces fonctions. Avec un système de calcul formel cette dérivation devient facile.

Exemple : en *Maple*, on calcule la dérivée par rapport à x de la façon suivante :

```
wint:=(x,y)->int (w(x,t),t=0..y):
uint:=(x,y,z)->int (u(x,t),t=y..z):
a:=(x,y,z)->diff(wint(x,y),x)+ uint(x,k(x),g(x)):
diff(a(x,y,z),x);
```

$$\int_0^y \frac{\partial^2}{\partial x^2} w(x,t)\,dt+\int_{k(x)}^{g(x)} \frac{\partial}{\partial x} u(x,t)\,dt$$
$$+u(x,g(x))\left(\frac{\partial}{\partial x}g(x)\right)-u(x,k(x))\left(\frac{\partial}{\partial x}k(x)\right)$$

Ci-avant, la syntaxe f :=(x,y) –> < *expression en x* et *y* > signifie que l'on définit une fonction de x et y appelée f.

Dans certains systèmes de calcul formel comme *Maple*, on peut même composer ou dériver des fonctions. La syntaxe est ici @ pour l'opérateur de composition des fonctions et D pour l'opérateur de dérivation.

Exemple :

```
f:=x->x^x:
(f@f)(x);
```

$$(x^x)^{(x^x)}$$

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

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

1.4 Simplification

Un des premiers problèmes auquel est confronté l'utilisateur d'un système de calcul formel lorsqu'il se met à traiter des expressions un peu complexes est celui de leur simplification. En effet, souvent, à la suite d'un calcul, l'utilisateur obtient un résultat compliqué qu'il veut simplifier. Il est alors confronté à l'un des problèmes les plus difficiles du calcul formel, celui de la simplification des expressions.

La notion de ce qui est simple est en elle-même mal définie et non absolue. De plus elle dépend de ce que l'on veut ensuite faire de l'expression.

Exemple : l'expression

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

semble beaucoup plus simple que la même expression développée

$$1 + x + x^2 + \ldots + x^{98} + x^{99}$$

mais, si l'on veut en calculer la primitive, la deuxième forme est plus facile à utiliser, c'est-à-dire qu'elle est plus simple pour ce calcul.

On peut en conclure qu'une fonction qui aurait pour but de « simplifier » dans le sens absolu du terme n'a pas grand sens dans un système de calcul formel. S'il existe en général de telles fonctions, il faut bien savoir quelles sont les simplifications qu'elles réalisent et, en fait, l'utilisateur dispose habituellement d'un ensemble de fonctions qui ont pour but tout à la fois de manipuler et de transformer une expression, et donc souvent de la simplifier.

Par conséquent, il faut se méfier des fonctions générales des systèmes de calcul formel du style `simplify` ; il vaut mieux utiliser des fonctions bien connues qui amènent sûrement à la simplification voulue.

De plus, la simplification des expressions demande en général, de la part de l'utilisateur, une certaine connaissance du système de calcul formel qu'il manipule, car il doit connaître l'éventail des fonctions qui sont à sa disposition.

Nous allons maintenant passer en revue les différents types d'expressions mathématiques que l'on peut rencontrer en indiquant à chaque fois ce que peut faire le calcul formel pour leur simplification.

1.4.1 Fonctions polynomiales

Le domaine des polynômes et des fractions rationnelles est celui pour lequel la simplification pose le moins de problèmes. Nous avons déjà vu un certain nombre d'opérations réalisées sur ces objets (§ 1.2). Celles-ci permettent de transformer une expression polynomiale et donc souvent de la simplifier.

Dans ce domaine, on peut avoir ce que l'on appelle des **formes canoniques** pour les polynômes et les fractions rationnelles, ce qui signifie une représentation unique. On peut donc toujours comparer de telles expressions afin de voir si elles sont égales et les simplifier en annulant les différences de termes égaux.

Il existe en général, dans tout système de calcul formel, une fonction qui offre la possibilité de mettre les fractions rationnelles sous une forme unique, par réduction au même dénominateur et normalisation du numérateur et du dénominateur. Ce n'est pas exactement une fonction de simplification, mais elle mène souvent à la simplification.

En *Maple*, cette fonction s'appelle `normal`.

Dans les deux exemples ci-après, elle permet de simplifier une expression, puis de vérifier la nullité d'une autre.

Exemple

```
e:=1+1/(1+1/(1+1/(1+1/(1+1/(1+1/(1+1/(1+1/(1+1/(1+1/
(1+1/(1+1/(1+1/(z+(1/(y+1/x))))))))))))))):
normal(e);
```

$$\frac{233zyx + 233z + 233x + 144yx + 144}{144zyx + 144z + 144x + 89yx + 89}$$

```
e:=(r^2+a^2+a)/(r^2+b^2-b)-(r^2+b^2+r^2+a^2+b+a)/
   (r^2+b^2+r^2+a^2-b-a)+
    (b-a)*(b+a)*(b^2+2*r^2-1+a^2)/
    (r^2+b^2-b)/(2*r^2+b^2+a^2-b-a);
```

$$e := \frac{r^2 + a^2 + a}{r^2 + b^2 - b} - \frac{2r^2 + b^2 + a^2 + b + a}{2r^2 + b^2 + a^2 - b - a} + \frac{(b-a)(b+a)(b^2 + 2r^2 - 1 + a^2)}{(r^2 + b^2 - b)(2r^2 + b^2 + a^2 - b - a)}$$

```
normal(e);
```

$$0$$

Il ne faut pas oublier non plus que certaines opérations, qui semblent très éloignées de la simplification, peuvent dans certains cas être utiles, comme le développement d'un polynôme qui permet de vérifier rapidement si ce dernier est nul.

La **factorisation** permet parfois aussi de simplifier, mais elle est très coûteuse et ne fonctionne pas dans tous les cas (§ 1.2).

Exemple : dans le cas multivariable suivant, la factorisation conduit à l'expression souhaitée :

```
v := -3*x**7*y**7*z**2-24*x**8*y**7*z**2+3*x**6*
   y**3*z**8*
w**2-18*x**9*y**4*z**2*w**2-2*x**6*y**3*z**3*
   w**3-14*x**6*
y**5*z**2*w**4+2*x**6*y**3*z**6*w**3+6*x**7*y**5*
   z**3*w+96*
x**8*y**6*z**5*w**2+58*x**7*y**3*z**3*w+42*x**7*
   y**7*z**2*
w**2+414*x**7*y**5*z**2*w**2-3*x**6*y**3*z**5*
   w**2-29*x**7*
y**5*z**2-36*x**7*y**4*z**8*w**2-24*x**7*y**4*
   z**6*w**3-8*
x**7*y**5*z**5*w**2-168*x**7*y**6*z**5*w**4+12*
   x**7*y**6*
z**5*w**2-216*x**10*y**5*z**5*w**2+18*x**9*y**4*
   z**5*w**2+
14*x**6*y**5*z**5*w**4+x**6*y**5*z**2*w**2-x**6*
   y**5*z**5*
w**2+87*x**7*y**3*z**5+9*x**7*y**5*z**5+522*
   x**10*y**4*z**2
-232*x**8*y**5*z**2+54*x**10*y**6*z**2 :
factor (v);
```

$$-z^2y^3x^6(-3xy^2 + w^2 - z^3w^2 - 29x + 12yz^3w^2x)(3z^3 + 2zw - y^2 - 8xy^2 + 14y^2w^2 + 18x^3y)$$

1.4.2 Fonctions algébriques

Lorsque la classe des expressions que l'on veut simplifier s'élargit, cela devient plus difficile.

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 ce que l'on

appelle une forme normale : l'expression est nulle si et seulement si sa forme normale est le symbole 0.

Après les fonctions polynomiales, on peut considérer la classe des fonctions algébriques qui sont les zéros de polynômes à coefficients dans $\mathbb{Q}$. Par exemple $\sqrt[n]{x}$ (zéro de $X^n - x$) et i (zéro de $X^2 + 1$) sont algébriques. Là, le problème de la simplification devient plus difficile car, selon le type d'expressions algébriques que l'on prend, on ne peut pas trouver de forme normale.

Sans forme normale, les systèmes ne peuvent que donner un certain nombre de fonctions de réécriture avec lesquelles l'utilisateur doit jongler pour parvenir à un résultat.

De plus, si l'on utilise une fonction de simplification assez générale, sans connaître exactement son rôle, le résultat peut être faux à cause de certaines assertions réalisées par cette fonction. On retrouve ce problème dans tous les systèmes de calcul formel.

Exemple : le cas suivant est typique : 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. Le système de calcul formel ne peut pas a priori savoir quel est le type de la variable x.

Dans ces cas, selon le système de calcul formel, il peut y avoir des résultats différents. En *Maple*, il laisse l'expression telle quelle, n'ayant pas d'autre information.

Dans certains systèmes de calcul formel, il est possible de dire quel est le type de la variable et même faire des hypothèses sur ses valeurs.

Dans l'exemple suivant, *Maple* ne simplifie pas dans le premier cas, mais après lui avoir dit que x était plus grand que r, il réalise la simplification désirée dans le second cas (les variables avec un « ~ » sont celles pour lesquelles une hypothèse a été faite) :

```
sqrt((x-r)^2);
```

$$\sqrt{(x-r)^2}$$

```
assume(x>r) :sqrt((x-r)^2);
```

$$x\tilde{} - r\tilde{}$$

Il existe cependant des possibilités de réaliser des simplifications lorsque l'on se trouve en présence d'expressions avec des radicaux qui ne sont pas trop imbriqués. Là encore il faut bien savoir utiliser l'éventail de fonctions fournies avec le système de calcul formel.

Exemple : pour simplifier des expressions où le dénominateur comprend des racines carrées comme :

$$\frac{2-x}{\sqrt{2}-\sqrt{x}}$$

c'est la fonction rationalize qu'il faudra utiliser en *Maple*, toutes les autres fonctions de simplification ne donnant rien :

```
rationalize((2-x)/(sqrt(2)-sqrt(x)));
```

$$\sqrt{x} + \sqrt{2}$$

1.4.3 Fonctions transcendantes

Les fonctions transcendantes sont celles qui ne sont pas algébriques.

Nous avons vu dans le paragraphe précédent (§ 1.4.2) que certaines classes d'expressions n'admettent pas une forme normale. Mais, 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 π et ln 2 et d'une variable, par utilisation répétée de l'addition, de la soustraction, du produit, de l'exponentielle et du sinus. 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.

Là encore, les systèmes de calcul formel ne donnent qu'un ensemble de fonctions qu'il va falloir connaître et manipuler avec habileté et précaution.

Souvent, certaines classes de fonctions bénéficient d'un ensemble de règles de réécriture qui permettent de réaliser certaines simplifications. C'est le cas pour les **fonctions trigonométriques**.

Exemple : on peut transformer les cosinus d'angles multiples en puissances de cosinus :

```
cos(4*arctan(x))+cos(6*arctan(x));
```

$$\cos(4\arctan(x)) + \cos(6\arctan(x))$$

```
normal(expand ("));
```

$$2\,\frac{1-10x^2+5x^4}{(1+x^2)^3}$$

Il ne faut pas oublier qu'il est parfois utile de transformer une expression pour pouvoir la simplifier.

Exemple : la simplification aveugle donne un résultat encore plus compliqué, alors qu'il suffit de remplacer l'arc tangente et le sinus hyperbolique par leurs expressions en fonction de logarithmes et d'exponentielles puis de simplifier :

```
v:=(exp(2*arctanh(1-x))-2*sinh(log(x))/(1-x))/3;
```

$$v := \frac{1}{3}\left(-e^{(-2\operatorname{arctanh}(-1+x))} - \frac{2}{3}\frac{\sinh(\ln(x))}{1-x}\right)$$

```
simplify(v);
```

$$\frac{1}{3}\left(-e^{(-2\operatorname{arctanh}(-1+x))} + e^{(-2\operatorname{arctanh}(-1+x))}x + 2\sinh(\ln(x))\right)/(-1+x)$$

```
normal(simplify(convert(e,expln),exp));
```

$$\frac{1}{x}$$

Le problème du type de la variable se pose à nouveau avec toutes les fonctions multiformes, comme le **logarithme** ou les **fonctions hypergéométriques**. Le calcul formel fournit alors peu d'assistance pour les simplifications. Là encore il faut se méfier des fonctions de simplification.

Exemple : une simple vérification numérique montre que la fonction simplify de *Maple* a commis une erreur à cause d'un choix incohérent de déterminations principales de fonctions multiformes. Ici, la fonction dilog est définie comme suit :

$$\operatorname{dilog}(x) = \int_1^x \frac{\ln(t)}{(1-t)}\,dt$$

```
e:=-I*dilog(-1/a)+I*ln(1/a)*ln(-1-a)-I*dilog(-a)
   +I*dilog(1/a);
```

$$e := -\mathbf{I}\,\operatorname{dilog}\left(-\frac{1}{a}\right) + \mathbf{I}\,\ln\left(\frac{1}{a}\right)\ln(-1-a) - \mathbf{I}\,\operatorname{dilog}(-a) + \mathbf{I}\,\operatorname{dilog}\left(\frac{1}{a}\right)$$

```
se:=simplify(e);
```

$$se := \frac{1}{2}\mathbf{I}\,\ln(-a)^2 - \mathbf{I}\,\ln(a)\ln(-1-a) + \mathbf{I}\,\operatorname{dilog}\left(\frac{1}{a}\right)$$

```
evalf(subs(a=2,e-se));
                -2.547612411+.310^-9 I
```

1.5 Courbes et surfaces

La visualisation des résultats est une étape essentielle dans la résolution d'un problème. Tout ingénieur est confronté à une telle tâche qui permet d'améliorer la compréhension de ce qui se passe et même parfois de résoudre le problème. C'est souvent aussi la seule façon de présenter les résultats.

Une caractéristique importante d'un système de calcul formel est la facilité avec laquelle on peut faire des tracés. En effet, le calcul formel manipule déjà les expressions mathématiques dans une syntaxe proche de la syntaxe habituelle et il ne reste plus qu'à utiliser la bonne fonction du système pour avoir la courbe ou la surface désirée. Ces fonctions dépendent bien évidemment du système de calcul formel, mais les fonctionnalités sont en général les mêmes avec plus ou moins de facilité pour modifier l'aspect du dessin, en particulier les points de vue lors du tracé de surfaces.

Nous allons passer en revue les divers types de tracés que l'on peut réaliser. Nous ne nous étendrons pas sur le choix des bornes de variation des variables, ni sur le nombre de points choisi pour faire le tracé. En général, les systèmes de calcul formel adaptent ce dernier en fonction de la courbe ou de la surface à tracer et de ses éventuelles discontinuités. Des outils sont aussi généralement disponibles dans les cas difficiles.

1.5.1 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 **une fonction paramétrée** $x = f(t)$, $y = g(t)$ ou **une fonction implicite** $f(x,y) = 0$ ou **une fonction en coordonnées polaires** $r = f(\theta)$. En *Maple*, tous ces tracés sont réalisables.

Exemple : tracer la courbe $y = (\sin(x))^2 - e^{-x}\cos(x)$ pour $x \in [0,10]$ se fait très simplement :

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

et le résultat est donné sur la figure **2**.

Le tracé d'une courbe paramétrée n'est pas plus difficile.

Exemple : on peut facilement superposer les courbes :

$$x(t) = -9\cos(t) - \cos(-9t),\ y(t) = -9\sin(t) - \sin(-9t)$$

et :

$$x(t) = 11\cos(t) - \cos(11t),\ y(t) = 11\sin(t) - \sin(11t).$$

Ici, il est plus facile d'écrire d'abord les fonctions $x(a,t)$ et $y(a,t)$.

```
x:=(a,t)->(1+a)*cos(t)–cos((1+a)*t):
y:=(a,t)->(1+a)*sin(t)–sin((1+a)*t):
plot([x(–10,t),y(–10,t),t=0..2*Pi],
      [x(10,t),y(10,t),t=0..2*Pi]);
```

Le résultat est donné sur la figure **3**.

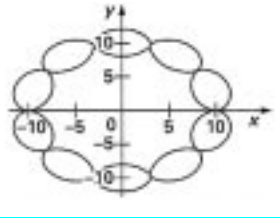

Figure 3 – Superposition de courbes paramétrées

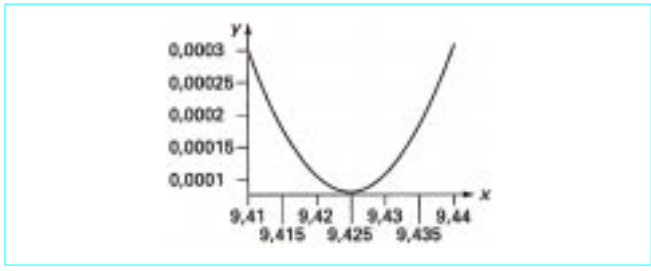

Figure 2 – Courbe $y = (\sin(x))^2 - e^{-x}\cos(x)$

Pouvoir tracer des courbes dont on n'a pas *a priori* l'expression explicite peut s'avérer être un outil très puissant, au détriment cependant de la précision du tracé. Cela est possible en *Maple*.

Exemple : on peut l'appliquer à la courbe :

$$x^6 + y^6 + 3x^4y^2 + 3x^2y^4 - x^4 + 2x^2y^2 - y^4 = 0.$$

```
plots[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);
```

Le résultat est donné sur la figure **4**.

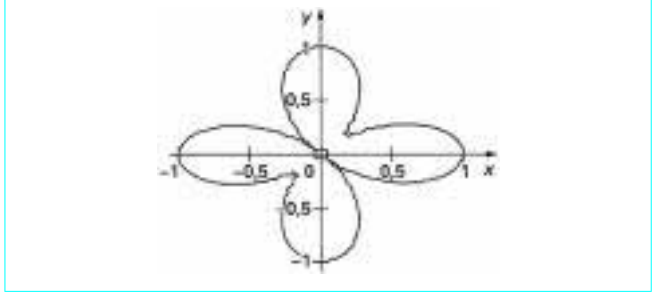

Figure 4 – Tracé de courbe implicite

Il est aussi possible de tracer des courbes à partir de la donnée d'un ensemble de points, venant, par exemple, de la sortie d'un programme numérique.

1.5.2 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 et les surfaces sont rendues par des maillages rectangulaires ou triangulaires avec élimination des parties cachées.

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 peut généralement être changée en utilisant des coordonnées sphériques ou bien, comme dans *Maple*, avec la souris en cliquant sur le tracé et en le faisant tourner. On peut aussi obtenir des aspects de perspective.

Ici encore le point essentiel est la facilité avec laquelle ces tracés peuvent être réalisés.

Exemple : le tracé de la surface $z = \cos(\sqrt{x^2 + y^2})$ d'un point de vue (55°, 20°) se fait immédiatement en *Maple* par :

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

Le résultat est donné sur la figure **5**.

On peut également tracer la courbe gauche

$$x(z) = \frac{\cos z}{z}, \qquad y(z) = \frac{\sin z}{z}$$

de la façon suivante :

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

Le résultat est donné sur la figure **6**.

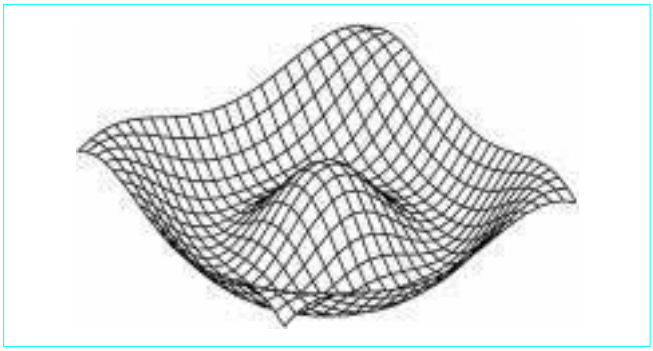

Figure 5 – Surface $z = \cos(\sqrt{x^2 + y^2})$

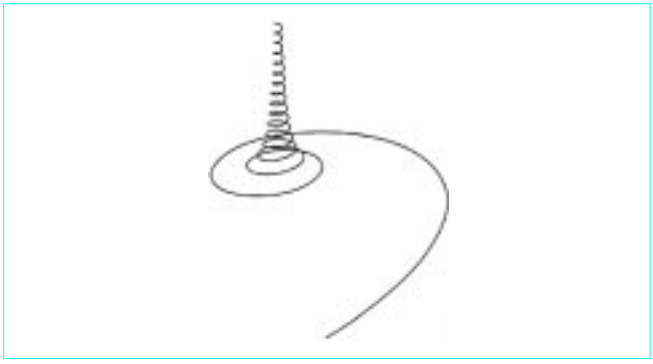

Figure 6 – Courbe gauche $x = \frac{\cos(z)}{z}$, $y = \frac{\sin(z)}{z}$

En *Maple*, on peut également tracer des données numériques, des lignes de niveau et aussi réaliser des tracés de surfaces implicites et des tracés d'intersection de surfaces.

Il faut bien sûr noter que tous ces tracés sont en couleurs, ce qui peut souvent faciliter la compréhension et l'analyse d'un dessin, surtout pour une surface.

1.5.3 Animation

Dans des systèmes de calcul formel comme *Maple* ou *Mathematica*, il existe aussi la possibilité de réaliser des animations, permettant ainsi de visualiser une quatrième dimension de nature plutôt temporelle. Cela consiste en l'affichage séquentiel de plusieurs tracés que l'on fait ensuite défiler à la façon d'un film.

2. Calcul intégral

Les intégrales et les primitives interviennent assez souvent dans les problèmes que l'ingénieur doit traiter. C'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 le calcul formel sait calculer sont bien délimitées.

Le but final du calcul intégral est très souvent l'intégration numérique. Pour cela, un grand nombre de programmes numériques performants réalisent ces opérations. On peut alors se demander quelle est l'utilité du calcul formel dans ce cas.

D'abord, le calcul, lorsqu'il est possible, de la primitive de l'expression à intégrer permet souvent de réduire les erreurs dues aux méthodes d'intégration numérique et d'éviter les problèmes d'instabilité.

Ensuite, si l'on doit calculer par exemple une intégrale du type :

$$\int_{a(x)}^{b(x)} f(x, y)\,\mathrm{d}y$$

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 numériquement plutôt que de faire chaque fois l'intégration numérique.

Nous allons décrire les classes de fonctions que le calcul formel sait intégrer. Nous commençons par les calculs de primitives, puis nous abordons les intégrales définies.

2.1 Calcul de 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.

Exemple : 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 comme *Maple* donne tout de suite une réponse :

```
int(1/x^3+1),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 le 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 théo-

rèmes 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 est défini plus loin (§ 2.1.2).

Nous allons passer en revue les différentes primitives que le calcul formel sait calculer.

2.1.1 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 $\mathbb{C}$, opération extrêmement coûteuse (voir § 1.2 et § 4.1). De plus, on va voir que cette factorisation n'est pas nécessaire à l'intégration.

En fait le calcul formel utilise la **propriété suivante** :

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 $K = \mathbb{Q}$, un système comme *Maple* parvient toujours au résultat et il devrait toujours y parvenir lorsque K est une extension algébrique de $\mathbb{Q}$.

Exemples

■ La décomposition en éléments simples de la fraction rationnelle :

$$\frac{1}{1+x^6}$$

ne donne pas ses pôles car ils ne sont pas rationnels, mais *Maple* sait en calculer la primitive :

```
int(1/(x^6+1),x);
```

$$\frac{1}{3}\arctan(x) - \frac{1}{12}\sqrt{3}\ln(x^2 - \sqrt{3}x + 1) + \frac{1}{6}\arctan(2x - \sqrt{3}) + \frac{1}{12}\sqrt{3}\ln(x^2 + \sqrt{3}x + 1) + \frac{1}{6}\arctan(2x + \sqrt{3})$$

■ Dans le cas suivant, le calcul de la primitive de :

$$\frac{x}{2 + x^2 + x^5}$$

fait intervenir des nombres algébriques de degré 5. *Maple* donne le résultat en fonction des racines d'un polynôme du cinquième degré dont on ne sait pas trouver une expression analytique des racines, mises sous la forme RootOf (...) :

```
int(x/(2+x^2+x^5),x);
```

$$\sum_{\_R = \%1} \_R \ln\left(x + \frac{50216}{25}\_R^4 - \frac{12554}{25}\_R^3 - \frac{18}{5}\_R^2 + \frac{1}{2}\_R + \frac{19}{50}\right)$$

$$\%1 := \text{RootOf}(25108\ \_Z^5 - 45\ \_Z^3 - 5\ \_Z^2 + 5\ \_Z + 1)$$

Le résultat précédent signifie que l'on fait la somme pour $\_R$ parcourant l'ensemble des racines du polynôme $25108\ \_Z^5 - 45\ \_Z^3 - 5\ \_Z^2 + 5\ \_Z + 1$ (polynôme en la variable $\_Z$). Cela montre, en particulier, qu'il n'y a pas d'expression analytique plus simple de la primitive.

2.1.2 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.

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 $x^2 + 1 = 0$.

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 (vers 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, cela signifie que si le système de calcul formel ne trouve pas une primitive élémentaire, c'est qu'il n'en existe pas.

L'implantation dans *Maple* est complète pour les extensions purement transcendantes, c'est-à-dire lorsqu'aucune fonction algébrique n'intervient. Sinon, certaines primitives sont trouvées par le système, mais il échoue parfois alors qu'une primitive élémentaire existe.

Exemples

■ Voici une fonction purement transcendante admettant une primitive élémentaire, *Maple* 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);
```

$$\frac{e^{(x^3+1)}}{x}$$

■ Voici une autre fonction purement transcendante et *Maple* ne trouve pas de primitive élémentaire, cela *prouve* donc qu'il n'en existe pas :

```
int(1/(ln(x^3+1)),x);
```

$$\int \frac{1}{\ln(x^3+1)}\,dx$$

À l'heure actuelle, le seul système de calcul formel qui implante complètement l'algorithme de Risch est *Axiom*.

2.1.3 Autres fonctions

Un certain nombre d'intégrales simples ne s'expriment pas à l'aide des fonctions élémentaires. Vu leur importance dans certaines applications, quelques-unes d'entre elles ont reçu un nom et font partie des **fonctions spéciales**. Les plus importantes sont la **fonction d'erreur,** l'**exponentielle intégrale,** le **sinus intégral** et le **cosinus intégral**, soit respectivement :

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \mathrm{e}^{-t^2}\,\mathrm{d}t, \qquad \mathrm{Ei}(n,x) = \int_1^\infty \frac{\mathrm{e}^{-xt}}{t^n}\,\mathrm{d}t$$

$$\mathrm{Si}(x) = \int_0^x \frac{\sin t}{t}\,\mathrm{d}t, \qquad \mathrm{Ci}(x) = \gamma + \ln(\mathrm{i}x) - \frac{\mathrm{i}\pi}{2} + \int_0^x \frac{\cos t - 1}{t}\,\mathrm{d}t.$$

Deux autres fonctions importantes aussi pour l'intégration sont la **fonction de Heaviside** qui est nulle pour les valeurs négatives et vaut 1 pour les valeurs positives, et sa dérivée la **fonction de Dirac**.

En général, les systèmes de calcul formel connaissent ces fonctions, c'est-à-dire 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. Ce qui signifie que, lorsque le système 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 : *Maple* peut calculer la primitive suivante :

```
f:=sin(x^2+1)/x;
```

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

```
int(f,x);
```

$$\frac{1}{2}\cos(1)\,\mathrm{Si}(x^2) + \frac{1}{2}\sin(1)\,\mathrm{Ci}(x^2)$$

Il faut aussi noter qu'il est souvent utile d'aider le système de calcul formel à trouver une primitive, en particulier, en utilisant l'intégration par parties ou bien des changements de variables. Cela peut parfois permettre de ramener la fonction dont on cherche la primitive dans une classe de fonctions que le système de calcul formel sait intégrer.

Exemple : la primitive de $\arcsin x$ est facilement obtenue par *Maple* (fonction élémentaire mais non purement transcendante) :

```
int(arcsin(x),x);
```

$$x\,(\arcsin x) + \sqrt{1-x^2}$$

mais *Maple* ne parvient pas à calculer celle de $(\arcsin x)^2$:

```
i:=int(arcsin(x)^2, x);
```

$$i := \int (\arcsin x)^2\,\mathrm{d}x$$

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 $(\arcsin x)^n$. On intègre deux fois par parties, en donnant comme arguments à la fonction *Maple* intparts du package student l'intégrale à calculer et la partie $u(x)$ à dériver de la formule d'intégration par parties :

$$\int u(x)v'(x)\,\mathrm{d}x = u(x)v(x) - \int u'(x)v(x)\,\mathrm{d}x.$$

```
student[intparts](i,arcsin(x)^2) ;
```

$$(\arcsin x)^2 x - \int 2\,\frac{(\arcsin x)x}{\sqrt{1-x^2}}\,\mathrm{d}x$$

```
student[intparts](",arcsin(x));
```

$$(\arcsin x)^2 x + 2(\arcsin x)\sqrt{1-x^2} + \int -2\,\mathrm{d}x$$

```
value(");
```

$$(\arcsin x)^2 x + 2(\arcsin x)\sqrt{1-x^2} - 2x$$

2.2 Intégrales définies

Pour calculer :

$$\int_a^b f(x)\,\mathrm{d}x$$

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, la plupart des systèmes de calcul formel savent calculer un certain nombre d'intégrales impropres autres que celles dont ils savent calculer la primitive.

Cette méthode n'est pas valide sans quelques hypothèses sur f. La vérification de ces hypothèses pose parfois de problème aux systèmes de calcul formel.

■ Une première difficulté est la **détection de singularités** dans l'intervalle d'intégration. Dans ce cas, le système de calcul formel doit les repérer sous peine d'obtention de résultats faux. Cela ne peut pas toujours marcher parfaitement et l'utilisateur doit être très prudent.

Dans les deux **exemples** suivants. *Maple* donne un résultat juste dans le premier cas, alors que, dans le second cas, il donne un résultat faux car il ne se rend pas compte que le dénominateur de la fonction s'annule deux fois. Ici, le tracé de la courbe correspondante peut aider à la recherche rapide des singularités :

```
int(1/x^2, x=-1..1);
```

$$\infty$$

```
f:=(2*x*exp(x)+x^2*exp(x)-2)/(x^2*exp(x)-2*x-1);
```

$$f := \frac{2x\,\mathrm{e}^x + x^2\mathrm{e}^x - 2}{x^2\mathrm{e}^x - 2x - 1}$$

```
int(f,x=-1..2);
```

$$\ln(4\mathrm{e}^2 - 5) - \ln(\mathrm{e}^{(-1)} + 1)$$

■ Une autre difficulté est le **choix de la bonne branche des fonctions multiformes**. Ce problème est aussi apparu dans le cas de la simplification (§ 1.1.3 et § 1.4.3) et il 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.

Exemple : dans le cas suivant, le résultat donné par *Maple* est faux (la vraie valeur est 0) :

```
f: :=1/(1+(x+2*I)^2);
```

$$f := \frac{1}{1 + (x + 2I)^2}$$

```
int(f,x=-infinity..infinity);
```

$$\pi$$

■ Outre les fonctions dont ils savent calculer la primitive, les systèmes de calcul formel peuvent, en général, calculer plusieurs classes d'intégrales définies. Cela dépend des systèmes.

Exemple : *Maple* sait calculer un certain nombre d'**intégrales elliptiques** ainsi que les intégrales d'autres fonctions comme dans le calcul suivant :

```
f:=exp(1-x^2)/x;
```

$$f := \frac{e^{(1-x^2)}}{x}$$

```
int(f,x=1..infinity);
```

$$\frac{1}{2}\, eEi\,(1, 1)$$

Le résultat est donné en fonction de l'exponentielle intégrale (§ 2.1.3) dont il est facile d'obtenir l'évaluation numérique.

2.2.1 Transformées intégrales

Les transformées intégrales sont un cas particulier d'intégrales impropres à un paramètre que les systèmes de calcul formel calculent en recherchant dans une table.

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.

Exemple : on veut trouver la fonction $f(t)$ solution de l'équation intégrale :

$$\sin t = 1 + \int_0^t e^x f(t-x)\, dx, \qquad t > 0.$$

On va utiliser pour cela la transformée de Laplace et sa transformée inverse.

En *Maple* cela donne :

```
eq:=sin(t)=1+int(exp(x)*f(t-x),x=0..t):
laplace (eq,t,s);
```

$$\frac{1}{s^2+1} = \frac{1}{s} + \frac{Laplace\,(f(t), t, s)}{s-1}$$

```
solve(",laplace(f(t),t,s));
```

$$\left(\frac{1}{s^2+1} - \frac{1}{s}\right)(s-1)$$

```
invlaplace(",s,t);
```

$$-Dirac\,(t) + 1 + \cos\,(t) - \sin\,(t)$$

La solution est donnée en fonction de la fonction de Dirac.

2.2.2 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 : pour calculer l'intégrale double :

$$\int\int (x^2 + y^2)\, dx\, dy$$

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);
```

$$\frac{4}{105}\frac{(ph)^{5/2}\sqrt{2}\,(15h + 14p)}{p^2}$$

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. En revanche, l'obtention d'une formule pour un ou deux niveaux d'intégration présente l'intérêt de pouvoir réduire considérablement la durée d'une évaluation numérique.

2.2.3 Intégration numérique

Tout système de calcul formel permet l'intégration numérique en utilisant les méthodes classiques des intégrateurs numériques. Cela est donc réalisable pour des intégrales que le calcul formel ne sait pas calculer.

Cependant, il faut être réaliste et savoir que, comme tout calcul numérique (§ 5), le temps de calcul est généralement long.

Mais le calcul numérique d'intégrales peut avoir son utilité, pour vérifier par exemple des calculs de primitives ou d'intégrales définies, ou dans le cours d'un long calcul formel.

3. Calcul matriciel

Le calcul matriciel est un domaine où le calcul formel peut apporter beaucoup. En effet, au-delà de matrices 3×3, 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.

La plupart du calcul matriciel en *Maple* se fait dans un package appelé `linalg`. Pour éviter d'utiliser le nom du package chaque fois que l'on utilise une fonction de celui-ci, il est possible de le charger une fois pour toutes à l'aide de la commande `with (linalg)`. C'est ce que nous supposerons dans tout le paragraphe.

3.1 Calculs de base

Tous les systèmes de calcul formel permettent de réaliser les calculs de base sur les matrices et les vecteurs qui sont la somme, le produit, l'élévation à une puissance et le calcul de l'inverse. Seule la syntaxe change d'un système de calcul formel à un autre.

Exemple : en *Maple*, pour créer une **matrice de Wandemonde** 3×3, un vecteur et en réaliser le produit, on fait :

```
m:=matrix(3,3,(i,j)->x.i^(j-1));
```

$$m := \begin{bmatrix} 1 & x1 & x1^2 \\ 1 & x2 & x2^2 \\ 1 & x3 & x3^2 \end{bmatrix}$$

```
v:=vector([1,2,3]);
```

$$v := [1\ 2\ 3]$$

```
evalm(m &*v);
```

$$[1 + 2x1 + 3x1^2 \qquad 1 + 2x2 + 3x2^2 \qquad 1 + 2x3 + 3x3^2]$$

Le troisième argument de la commande matrix est une fonction qui crée les éléments de la matrice et la commande evalm permet d'évaluer une expression matricielle. L'opérateur « . » est l'opérateur de concaténation de chaînes de caractères et la multiplication matricielle est exprimée par &*.

On calcule de même facilement l'inverse de cette matrice que l'on factorise élément par élément (c'est la fonction map qui permet d'appliquer la fonction factor à tous les éléments de la matrice), ainsi que le déterminant factorisé :

```
map(factor,evalm(m^(-1)));
```

$$\left[\frac{x3x2}{(-x3+x1)(-x2+x1)}, -\frac{x1x3}{(-x3+x2)(-x2+x1)}, \frac{x1x2}{(-x3+x2)(-x3+x1)}\right]$$

$$\left[-\frac{x2+x3}{(-x3+x1)(-x2+x1)}, \frac{x1+x3}{(-x3+x2)(-x2+x1)}, -\frac{x1+x2}{(-x3+x2)(-x3+x1)}\right]$$

$$\left[\frac{1}{(-x3+x1)(-x2+x1)}, -\frac{1}{(-x3+x2)(-x2+x1)}, \frac{1}{(-x3+x2)(-x3+x1)}\right]$$

```
factor(det(m));
```

$$-(-x3+x2)(-x3+x1)(-x2+x1)$$

Il faut noter que ces calculs ne sont limités que par la capacité de la machine utilisée en place mémoire et en temps de calcul, et qu'il ne faut pas espérer calculer l'inverse d'une matrice pleine au-delà d'une dimension 10×10.

3.2 Résolution de systèmes linéaires

Une des applications importantes du calcul matriciel est la résolution d'un système linéaire d'équations de la forme $Ax = b$. Dans la plupart des applications, la matrice A est numérique et de grande taille. Pour de gros systèmes, il n'existe pas d'autre recours que les techniques du calcul numérique qui permettent de traiter des systèmes denses de taille 100×100 et des systèmes creux de taille $1\,000 \times 1\,000$. 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.

En général, les systèmes de calcul formel permettent de résoudre les systèmes linéaires donnés sous la forme d'équations ou sous forme matricielle.

C'est sous cette dernière forme que l'on traite l'exemple suivant en *Maple* :

```
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]);
```

$$b := [t \quad -t \quad 1 \quad 4]$$

```
linsolve(A,b);
```

$$\left[-\frac{1}{6}\frac{2t+15}{t} \qquad -\frac{1}{6}\frac{8t+15}{t} \qquad \frac{1}{3}\frac{5t^2+57t+135}{(2t+9)t} \qquad \frac{1}{3}\frac{5t^2+48t+135}{(2t+9)t}\right]$$

Une méthode, que l'on n'emploie jamais en calcul numérique, consiste à calculer l'inverse de la matrice A avant de réaliser le produit $A^{-1}b$. Le principal intérêt de ce calcul relativement coûteux est qu'il réduit les résolutions ultérieures à un simple produit de matrices. Cette méthode peut être préférable dans le cas de nombreuses résolutions correspondant à différents vecteur $\vec{b}$.

Il est possible d'appliquer cette méthode à l'exemple précédent.

```
IA::=evalm(A^(-1));
```

$$IA := \left[\frac{-1}{6}, -\frac{1}{6}\frac{t+3}{t}, -\frac{1}{6}\frac{t+3}{t}, -\frac{1}{6}\frac{t+3}{t}\right]$$

$$\left[-\frac{1}{6}\frac{t+3}{t}, \frac{-1}{6}, -\frac{1}{6}\frac{t+3}{t}, -\frac{1}{6}\frac{t+3}{t}\right]$$

$$\left[\frac{1}{6}\frac{t+3}{t}, \frac{1}{6}\frac{t+3}{t}, \frac{1}{3}\frac{9t+27+t^2}{t(2t+9)}, \frac{1}{3}\frac{12t+27+t^2}{t(2t+9)}\right]$$

$$\left[\frac{1}{6}\frac{t+3}{t}, \frac{1}{6}\frac{t+3}{t}, \frac{1}{3}\frac{12t+27+t^2}{t(2t+9)}, \frac{1}{3}\frac{9t+27+t^2}{t(2t+9)}\right]$$

```
evalm(IA &*b);
```

$$\left[-\frac{1}{6}\frac{2t+15}{t} \qquad -\frac{1}{6}\frac{8t+15}{t} \qquad \frac{1}{3}\frac{5t^2+57t+135}{(2t+9)t} \qquad \frac{1}{3}\frac{5t^2+48t+135}{(2t+9)t}\right]$$

La matrice inverse ainsi obtenue peut être, par exemple, transformée en code Fortran et utilisée dans des programmes numériques (§ 5.2).

3.3 Calcul de valeurs propres

Le calcul de déterminants symboliques permet celui de polynômes caractéristiques. La connaissance de ce polynôme fournit de nombreuses informations sur la matrice, en particulier sur sa diagonalisation.

Le calcul de ce polynôme pour la matrice de l'exemple précédent (§ 3.2) est très facile en utilisant un système de calcul formel. On peut en calculer le discriminant, qui donne des informations sur la multiplicité des racines. En effet, la condition pour qu'un polynôme n'ait pas de racine multiple est qu'il n'ait pas de racine commune avec sa dérivée. Cette information est donnée par le résultant du polynôme et du polynôme dérivé, appelé le **discriminant** (§ 4.1).

Exemple :

```
charpoly(A,lambda);
```

$$108t^2 + 24t^3 + 81\lambda^2 - 216\,t\,\lambda - 36\,\lambda\,t^2 - 24\,t\,\lambda^2 + 18\,\lambda^3 + \lambda^4 - 4\lambda^2 t^2$$

```
factor(discrim(",lambda));
```

$$768\,t^4\,(t+3)^4\,(4t+9)^2\,(8t+27)$$

De cela il découle que, pour $t \notin \{-27/8, -3, -9/4, 0\}$, le polynôme caractéristique n'a que des racines simples et donc que la matrice est diagonalisable pour toutes ces valeurs de t.

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 (§ 4.1).

Exemple : dans le cas de la matrice précédente (§ 3.2), le calcul des valeurs propres ne pose pas de problème :

```
eigenvals(A);
```

$$2t, -2t-9, -\frac{9}{2} - \frac{1}{2}\sqrt{81+24t}, -\frac{9}{2} + \frac{1}{2}\sqrt{81+24t}$$

Indépendamment de l'utilisation des valeurs propres pour un grand nombre de problèmes qui apparaissent en physique et en automatique par exemple, l'obtention de la **forme de Jordan** d'une matrice a des applications pour le calcul des puissances et de l'exponentielle d'une matrice. 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.

L'**exponentielle d'une matrice** intervient lors de la résolution de systèmes différentiels linéaires à coefficients constants. Un tel système s'écrit :

$$\frac{\mathrm{d}X(t)}{\mathrm{d}t} = AX(t), \qquad X(0)\text{donné},$$

où $X(t)$ est un vecteur $(x_1(t), ..., x_n(t))$ d'ordre n et A est une matrice $n \times n$ indépendante de t. L'unique solution de ce système est donnée par :

$$X(t) = \mathrm{e}^{tA}X(0).$$

L'exponentielle de la matrice est définie par son développement en série :

$$\mathrm{e}^{tA} = \sum_{k=0} \frac{t^k A^k}{k!}$$

avec $A^0 = I$ la matrice identité.

La forme de Jordan intervient lors du calcul des A^k.

En général, les systèmes de calcul formel ont une fonction qui permet de calculer directement l'exponentielle d'une matrice.

Exemple : on peut appliquer cela à la résolution du système différentiel suivant :

$$\begin{cases} x'(t) = 3\,y(t) - z(t) \\ y'(t) = x(t) - y(t) + 2z(t) \\ z'(t) = x(t) + z(t) \end{cases}$$

```
A:=matrix([[0,3,-1],[1,-1,2],[1,0,1]]):
exponential(A,t);
```

$$\left[\frac{2}{3}\,\mathrm{e}^{(-t)} + \frac{1}{3}\,\mathrm{e}^{(2t)}, \frac{1}{3}\,\mathrm{e}^{(2t)} - \frac{1}{3}\,\mathrm{e}^{(-t)} + 2\,te^{(-t)}, \frac{1}{3}\,\mathrm{e}^{(2t)} - \frac{1}{3}\,\mathrm{e}^{(-t)} - 2\,te^{(-t)}\right]$$

$$\left[\frac{1}{3}\,\mathrm{e}^{(2t)} - \frac{1}{3}\,\mathrm{e}^{(-t)}, \frac{2}{3}\,\mathrm{e}^{(-t)} + \frac{1}{3}\,\mathrm{e}^{(2t)} - te^{(-t)}, \frac{1}{3}\,\mathrm{e}^{(2t)} - \frac{1}{3}\,\mathrm{e}^{(-t)} + te^{(-t)}\right]$$

$$\left[\frac{1}{3}\,\mathrm{e}^{(2t)} - \frac{1}{3}\,\mathrm{e}^{(-t)}, \frac{1}{3}\,\mathrm{e}^{(2t)} - \frac{1}{3}\,\mathrm{e}^{(-t)} - te^{(-t)}, \frac{2}{3}\,\mathrm{e}^{(-t)} - \frac{1}{3}\,\mathrm{e}^{(2t)} + te^{(2t)}\right]$$

Et on obtient la matrice qui donne la solution du système.

3.4 Applications

Les deux petites applications ci-après rendent compte de l'utilité pratique du calcul formel dans le domaine du calcul matriciel.

3.4.1 Automatique

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 $(f_1(X), ..., f_m(X))$ par rapport au vecteur $(X_1, ..., X_n)$ est la matrice dont l'élément (i, j) est $(\partial f_i / \partial X_j)$.

Exemple : pour fixer les idées voici les calculs mis en jeu pour le système consistant en un pendule inversé sur un chariot mobile (figure 7). Les équations du pendule sont les suivantes :

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = [u_1 + \sqrt{m_b}\,(\sin(x_3)x_4^2 - \cos(x_3)D_4)]/(m_b + m_c) \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = D_4 \end{cases} \quad \begin{cases} y_1 = x_1 \\ y_2 = x_3 \end{cases}$$

avec :

$$\begin{cases} \mathrm{D}_4 = -\frac{1}{d}\left[\sin(x_3)\cos(x_3)q_m x_4^2 + \frac{2}{dm_b\ell}\sin(x_3)m_b g - q_m\cos(x_3)u_1\right] \\ q_m = m_b / (m_b + m_c) \\ d = \frac{4}{3} - q_m\cos(x_3)^2 \end{cases}$$

où x_1, x_2, x_3 et x_4 correspondent respectivement à la position du chariot, à sa vitesse, à l'angle θ du pendule avec la verticale et à sa vitesse angulaire, m_b et m_c sont respectivement les masses du pendule et du chariot, ℓ est la longueur du pendule, g est l'accélération due à la pesanteur. 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*1)*
   (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])));
```

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -6\dfrac{g\sqrt{mb}}{(mb+4mc)\ell} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 6\dfrac{g(mb+mc)}{(mb+4mc)\ell} & 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \dfrac{+4\ell mc+6\sqrt{mb}}{(\ell+4mc)(mb+mc)} \\ 0 \\ \dfrac{1}{(mb+4mc)\ell} \end{bmatrix},$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

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.

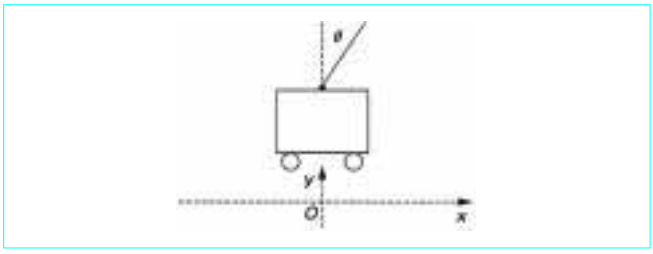

Figure 7 – Pendule inversé sur un chariot mobile

3.4.2 Mécanique

La mécanique a été l'un des premiers domaines de prédilection pour l'utilisation du calcul formel, en particulier la mécanique céleste où les calculs sont très rapidement inextricables. Il existe d'ailleurs des systèmes de calcul formel spécialisés pour ce domaine.

L'exemple le plus frappant et le plus célèbre est celui du calcul du mouvement de la Lune par Delaunay au XIX^e siècle qui lui a demandé plus de 10 ans pour une formule de 138 pages. Actuellement, le même calcul se réalise en quelques minutes.

Nous allons traiter ici un exemple qui montre déjà la difficulté des problèmes mécaniques, même dans un cas simple.

Un autre exemple plus complexe, en mécanique, faisant intervenir les équations de Lagrange, est donné dans le paragraphe 5.3.

Exemple : nous souhaitons calculer la matrice d'une rotation de l'espace à trois dimensions donnée par un vecteur directeur $\vec{u}$ unitaire de son axe et par son angle α 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));
```

$$R := [\cos(\alpha) + (1-\cos(\alpha))\,p^2,\ -\sin(\alpha)\,r + (1-\cos(\alpha))\,p\,q,\ \sin(\alpha)q + (1-\cos(\alpha))\,p\,r]$$

$$[\sin(\alpha)r + (1-\cos(\alpha))\,p\,q,\ \cos(\alpha) + (1-\cos(\alpha))\,q^2,\ -\sin(\alpha)p + (1-\cos(\alpha))\,q\,r]$$

$$[-\sin(\alpha)q + (1-\cos(\alpha))\,p\,r,\ \sin(\alpha)p + (1-\cos(\alpha))\,q\,r,\ \cos(\alpha) + (1-\cos(\alpha))\,r^2]$$

Mais, en dimension 3, on utilise souvent les coordonnées sphériques. Le vecteur unitaire $\vec{u}$ s'écrit 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 nouvelle matrice R :

```
col(subs([p=sin(phi)*cos(theta),
   q=sin(phi)*sin(theta), r=cos(phi)],"),1);
```

$$\begin{aligned}[\cos(\alpha) &+ (1-\cos(\alpha))\,\sin(\phi)^2\cos(\theta)^2\sin(\alpha)\cos(\phi) \\ &+ (1-\cos(\alpha))\,\sin(\phi)^2\cos(\theta)\sin(\theta) \\ &- \sin(\alpha)\sin(\phi)\sin(\theta) \\ &+ (1-\cos(\alpha))\,\sin(\phi)\cos(\theta)\cos(\phi)]\end{aligned}$$

Même si l'on connaît l'axe de la rotation et que la seule inconnue est son angle α, l'expression reste compliquée, surtout si l'on se met à composer des rotations d'axes différents.

4. Résolution d'équations

La résolution d'équations ou de systèmes d'équations est au cœur de la plupart des problèmes qui se posent à l'ingénieur. Dans ce domaine, la calcul formel peut jouer un rôle non négligeable.

4.1 Équations non linéaires

Nous avons déjà vu (§ 3.2) la résolution des systèmes d'équations linéaires. Nous parlons ici des équations et des systèmes d'équations non linéaires.

Dans tout système de calcul formel existe généralement une fonction qui permet de résoudre ces types de problèmes ; en *Maple*, elle s'appelle solve.

Cette fonction permet de résoudre les équations polynomiales de façon exacte jusqu'au degré 4. Si l'on a de la chance, selon le type de polynôme, on peut les résoudre pour des degrés supérieurs, même si le polynôme est irréductible comme celui de l'exemple ci-après.

$4x^8 + 48x^7 + 256x^6 + 792x^5 + 1\,590x^4 + 2\,196x^3 + 2\,104x^2 + 1\,290x + 459$

Exemple : *Maple* résout ce polynôme sans problème :

```
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);
```

$$-\frac{3}{2}+\frac{1}{2}\sqrt{-1+2\sqrt{-18+\sqrt{13}}},$$

$$-\frac{3}{2}-\frac{1}{2}\sqrt{-1+2\sqrt{-18+\sqrt{13}}},$$

$$-\frac{3}{2}+\frac{1}{2}\sqrt{-1-2\sqrt{-18+\sqrt{13}}},$$

$$-\frac{3}{2}-\frac{1}{2}\sqrt{-1-2\sqrt{-18+\sqrt{13}}},$$

$$-\frac{3}{2}+\frac{1}{2}\sqrt{(-1+2)\sqrt{-18-\sqrt{13}}},$$

$$-\frac{3}{2}-\frac{1}{2}\sqrt{-1+2\sqrt{-18-\sqrt{13}}},$$

$$-\frac{3}{2}+\frac{1}{2}\sqrt{-1-2\sqrt{-18-\sqrt{13}}},$$

$$-\frac{3}{2}-\frac{1}{2}\sqrt{-1-2\sqrt{-18-\sqrt{13}}}$$

Mais, en règle générale, les solutions exactes de polynômes de degré supérieur à deux présentent peu d'intérêt. Il vaux mieux se demander ce que l'on cherche à faire avec ces solutions, et un calcul de résultant ou une résolution numérique permettent souvent d'aboutir au résultat.

C'est ce qui a été fait dans un exemple (§ 1.2).

Lorsque l'on a un système d'équations non linéaires, le problème de la résolution devient beaucoup plus difficile. Dans le cas où l'on a des équations polynomiales, on peut utiliser le calcul de résultants si l'on a deux équations (§ 1.2).

Mais il existe un outil puissant qui permet d'aider à la résolution de ces équations, ce sont les **bases de Gröbner** ou bases standards. Le but n'est pas ici de décrire exactement cet outil, il existe pour cela des livres plus détaillés ; nous ne donnons qu'un exemple d'utilisation.

Principe : en partant d'un système de polynômes, on peut trouver (ou déduire) d'autres polynômes plus simples qui s'annulent aux mêmes points que ces polynômes. On obtient ainsi un système de polynômes plus simples (du point de vue du degré des polynômes). comme il y a généralement plusieurs monômes de chaque degré dans le système de départ, il faut choisir dans quel ordre on décide de les annuler. Une **base de Gröbner** est un système déduit du système de départ, où l'on ne peut plus effectuer aucune réduction compte tenu de l'ordre choisi. Cette base est unique.

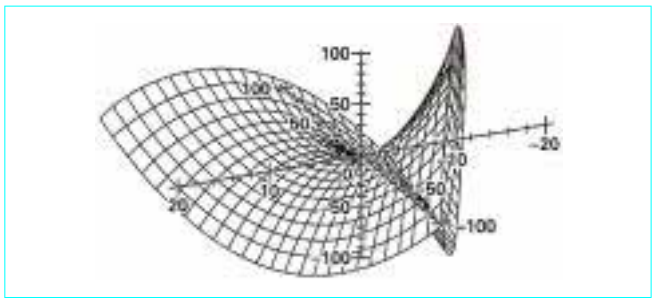

Figure 8 – Surface $x = uv$, $y = u + v$, $z = u^2 - v^2$

Cet outil permet alors éventuellement d'éliminer des variables et de répondre aux questions suivantes :

— le système d'équations a-t-il une solution (sur $\mathbb{C}$) ? ;

— le système a-t-il un nombre fini de solutions (sur $\mathbb{C}$) ?

Les deux fonctions *Maple* solvable et finite du package grobner permettent de répondre à cette question, alors que, lorsque la fonction générale de résolution de systèmes ne trouve aucune solution, on ne sait pas s'il n'y en a pas ou bien si elle n'en a pas trouvé.

Exemple : le système d'équations suivant a un nombre fini de solutions, mais il n'en a plus aucune si l'on ajoute une équation :

```
sys:={x^2+y^2+z^2+1,x^2+2*y^2-y*z-1,x+z^3-1};
```

sys := $\{x^2+y^2+z^2+1, x^2+2y^2-yz-1, x+z^3-1\}$

```
grobner[finite] (sys);
```

true

```
sys:=sys union {x^3+2*x*y-2};
```

sys : = $\{x^2 + y^2 + z^2 + 1,\ x^2 + 2y^2 - yz - 1,\ x^3 + 2xy - 2,\ x + z^3 - 1\}$

```
grobner[solvable](sys);
```

false

Mais cela ne permet pas de trouver la ou les solutions.

L'élimination d'une ou plusieurs variables d'un système d'équations polynomiales a des applications pratiques en géométrie.

Exemple : la surface représentée sur la figure **8** a été tracée grâce à la représentation paramétrique :

$$x = uv, \quad y = u + v, \quad z = u^2 - v^2.$$

On peut obtenir une relation entre x, y et z ainsi, où plex correspond à l'ordre choisi (ici purement lexicographique sur les variables) :

```
sys:={x-u*v,y-(u+v),z-(u^2-v^2):}
grobner[gbasis](sys,[u,v,x,y,z],plex);
```

$[-y + u + v,\ 2v^2 - y^2 + z + 2x,\ z - y^2 + 2vy,$
$-yz + y^3 - 4xy + 2vz,\ z^2 - y^4 + 4y^2x]$

Le dernier polynôme de cette base (et lui seul) ne fait apparaître ni u, ni v, et donne donc l'équation cherchée.

Il faut savoir aussi que ces calculs de bases de Gröbner sont très coûteux.

4.2 Équations différentielles

La résolution des équations différentielles ordinaires est un domaine majeur du calcul scientifique. Ici aussi, les résolutions sont essentiellement numériques et de nombreuses méthodes ont été développées pour en rechercher des solutions approchées par le calcul numérique et il en existe un grand nombre de très performantes. Mais ces méthodes se dégradent généralement loin des conditions initiales ou en présence de singularités. En revanche, bien que couvrant un champ beaucoup plus restreint, le calcul formel, par la recherche de solutions exactes ou sous la forme de développements en série, permet parfois d'éliminer ce défaut.

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. Il en va de même pour les équations aux dérivées partielles, qui sortent encore du champ d'application du calcul formel.

4.2.1 Solutions exactes

La recherche de solutions exactes, c'est-à-dire exprimées en termes de fonctions élémentaires, 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 de calcul formel à l'autre. Cependant, les équations suivantes sont généralement reconnues (y est la fonction inconnue, x la variable de dérivation) :

— **équations linéaires** :

$$a_0(x)y^{(p)}(x) + a_1(x)y^{(p-1)}(x) + \ldots + a_p(x)y(x) = b(x)\ ;$$

— **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** :

$$a_0x^ny^{(n)}(x) + a_1x^{n-1}y^{(n-1)}(x) + \ldots + a_ny(x) = 0.$$

Les systèmes de calcul formel possèdent des procédures spécialisées qui leur permettent de simplifier ces équations, parfois jusqu'à la solution. Cela ne dispense cependant pas l'utilisateur de connaître ces équations, car, lorsqu'un système n'arrive pas à les résoudre, on ne bénéficie pas de ses efforts intermédiaires pour les simplifier. 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.

Exemple : l'équation est autonome :

$$(y'')^2 = (1 + y'^2)^3,\ y(0) = y'(0) = 1.$$

En *Maple*, la résolution est donnée par l'utilisation de la fonction dsolve. Noter que l'on donne à la fonction les conditions initiales, sinon des constantes apparaissent dans la solution.

```
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));
```

$$y(x) = I\sqrt{x^2 + x\sqrt{2} - \frac{1}{2}} + \frac{1}{2}\sqrt{2} + 1,$$

$$y(x) = -I\sqrt{x^2 - x\sqrt{2} - \frac{1}{2}} - \frac{1}{2}\sqrt{2} + 1$$

Ici encore, comme pour le calcul d'intégrales, lorsque le système n'arrive pas à résoudre l'équation différentielle, l'utilisateur doit l'aider pour éventuellement ramener cette équation dans une classe connue par le système.

Exemple : *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. En *Maple*, cela se fait de la façon suivante :

```
subs(y(x)=Y(ln(x)),eq):
subs({ln(x)=t,x=exp(t)},");
```

$$\frac{D^{(2)}(Y)(t)}{(e^t)^2} - \frac{D(Y)(t)}{(e^t)^2} = \frac{Y(t)D(Y)(t)}{(e^t)^2}$$

```
map(numer,");
```

$$D^{(2)}(Y)(t) - D(Y)(t) = Y(t)D(Y)(t)$$

```
dsolve(",Y(t));
```

$$t = 2\,\frac{\arctan\left(\frac{1}{2}\,\frac{2Y(t)+2}{\sqrt{2\_C1-1}}\right)}{\sqrt{2\_C1-1}} - \_C2$$

Il ne reste plus qu'à effectuer le changement de variable inverse :

```
subs(t=ln(x),solve(",Y(t)));
```

$$\left(\tan\left(\frac{1}{2}\ln(x)\sqrt{2\_C1-1} + \frac{1}{2}\,\_C2\sqrt{2\_C1-1}\right) - \frac{1}{\sqrt{2\_C1-1}}\right)\sqrt{2\_C1-1}$$

ce qui montre que les solutions de l'équation différentielle sont de la forme

$$y = a\tan(a\ln(x)/2 + b) - 1.$$

Les systèmes de calcul formel peuvent aussi résoudre des systèmes d'équations différentielles, principalement les systèmes linéaires.

On a déjà vu une méthode pour résoudre les systèmes d'équations différentielles linéaires du premier ordre (§ 3.3).

4.2.2 Développements en séries

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 pour certains calculs asymptotiques. Dans ce domaine, on peut donner l'exemple suivant.

Exemple : on considère l'équation :

$$y' = \sin(x) + xy^2(x), \quad y(0) = 1,$$

ce qui se traduit en *Maple* par :

```
deq:=diff(y(x),x)=sin(x)+x*y(x)^2:
```

La courbe correspondante est représentée sur la figure **9**. Il y apparaît un comportement singulier vers x ≈ 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 ρ. 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 **9** 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 ρ. Cela 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,");
```

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

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.

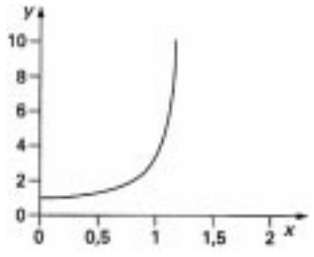

Figure 9 – Solution de ($y' = \sin(x) + xy^2(x)$, $y(0) = 1$)

4.2.3 Méthodes numériques

Il est très difficile de résoudre une équation différentielle explicitement, mais il existe de nombreuses méthodes numériques pour 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 formel et que l'on veuille poursuivre une étude numérique. 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 est 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), généralement aux extrémités du domaine d'étude. Les systèmes de calcul formel proposent habituellement diverses méthodes de résolution pour ces problèmes, dont la très classique méthode de Runge-Kutta.

Nous allons voir comment cela se passe en *Maple*.

Exemple : on considère l'équation :

$$y' = \sin(xy), \; y(0) = 1.$$

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. Le résultat de la commande *Maple* dsolve est ici une fonction $f(x)$ que l'on peut utiliser pour calculer des valeurs de la solution :

```
f:=dsolve({diff(y(x),x)=sin(x*y(x)),y(0)=1},y(x),
   numeric):
f(0.5),f(1);
```

$$[x = .5,\ y(x) = 1.12970777182128090],$$

$$[x = 1,\ y(x) = 1.53409989659743085]$$

On peut aussi obtenir le tracé de la solution d'une équation différentielle. En *Maple*, on utilise une commande spéciale de tracé de courbe.

Exemple : si l'on veut tracer la solution de l'équation différentielle :

$$y'(x) = x \sin x \, e^{-|y(x)|}, \quad y(0) = 1,$$

on fait :

```
deq:=diff(y(x),x)=x*sin(x)*exp(—abs(y(x))):
f:=dsolve({deq,y(0)=1},y(x),numeric):
plots[odeplot](f,y(x),0..20);
```

La courbe obtenue est donnée sur la figure **10**.

5. Calcul numérique

Parfois, on parle de calcul formel par opposition au calcul numérique, alors que c'est la collaboration entre les deux qui doit permettre d'améliorer la résolution des problèmes dans le domaine du calcul scientifique. On peut dire que le calcul numérique est souvent l'aboutissement du calcul formel, ou bien considérer ce dernier comme un préprocesseur pour le calcul numérique. C'est dire si les liens qui existent entre les deux, qui d'ailleurs se développent de plus en plus, sont d'une extrême importance.

Nous verrons dans ce paragraphe l'utilisation du calcul numérique à l'intérieur d'un système de calcul formel, puis comment faire jouer à un système de calcul formel ce rôle de préprocesseur, par la génération de code et par des liens avec les systèmes numériques.

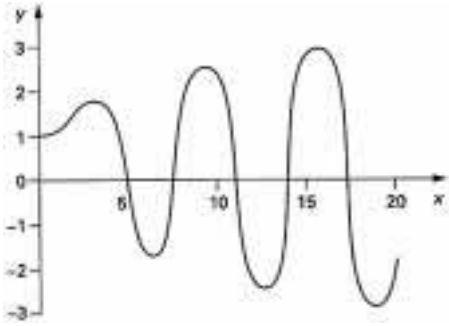

Figure 10 – Solution de $(y'(x) = x \sin x e^{-|y(x)|}, y(0) = 1)$

Pour les exemples en *Maple*, nous supposons encore ici le package linalg définitivement chargé (voir le début du § 3).

5.1 Calcul numérique dans un système de calcul formel

Nous avons déjà vu (§ 1.1.4) que l'on pouvait faire du calcul numérique dans un système de calcul formel à l'aide de nombres flottants avec un nombre de chiffres déterminé défini par l'utilisateur, ce qui permet des calculs avec une grande précision.

Intéressons-nous maintenant au temps de calcul. En effet, la principale faiblesse des systèmes de calcul formel en calcul numérique est leur lenteur. Prenons, par exemple, un calcul classique en analyse numérique qui est celui des valeurs singulières d'une matrice. Les valeurs singulières d'une matrice A sont les racines carrées des valeurs propres de $A^t A$. Le rapport entre la plus grande valeur singulière et la plus petite, le conditionnement de la matrice, quantifie la sensibilité de la solution x du système linéaire $A\,x = b$ par rapport aux données A et b. La solution est d'autant plus sensible que le conditionnement est grand.

Exemple : calculons les valeurs singulières d'une matrice numérique 100×100 obtenue aléatoirement en *Maple* :

```
m:=evalm(randmatrix(100,100)/99.0):
time(evalf(Svd(m)));
                              31.966
```

Le résultat est obtenu après 32 secondes sur une machine DecAlpha 300. Sauvegardons dans un fichier les éléments de la matrice utilisée et réalisons le même calcul dans un système spécialisé comme *Scilab* (§ 5.3) qui exécute en fait des programmes Fortran :

```
-->m=read('mat100.data',100,100);
-->initimer(),v=svd(m) ; timer()
   ans    =
0.6
```

On peut vérifier que les résultats donnés par les deux systèmes sont les mêmes, mais le temps de calcul est de 32 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.

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 avec une précision arbitraire. Mais ces structures plus générales sont plus lourdes à manipuler, d'où un plus grand nombre d'opérations élémentaires.

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.

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, que le temps de calcul ne soit pas une contrainte impérative.

Exemple : 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)));
                              47.366
v[1],v[15],v[16],v[17],v[30];
1.986492568608736347453487131171422755538\
      76938069200, .271123981808233807717 6\
      49933449653667978713669517601 0^-14, .9\
      82963192235731737522290562292476303\
      6335350305878510^-16, .319144776259661\
      50949774017342529486456377251924053 10^-17,
      .46986365445013202024187188936355 71\
      153148836611325810^-43
```

Le résultat s'obtient en 48 secondes, et quelques valeurs singulières sont affichées ci-avant, 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) et 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 lorsque l'on prend des flottants de 16 chiffres 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.

Il faut cependant noter qu'il est aussi possible, pour un système de calcul formel, d'utiliser des flottants machines ; cela accélère l'exécution des calculs, en particulier pour le tracé des courbes (calcul des coordonnées des points). Mais, on n'obtient pas la vitesse des langages numériques à cause de la structure des données.

5.2 Génération de code

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, car il ne commet pas d'erreur dans les calculs de dérivées et fournit des optimisations non négligeables.

Depuis un système de calcul formel, il est donc important de savoir produire du code numérique. Il y a deux moyens pour cela :

— **la traduction d'une expression** : l'utilisateur insère les expressions traduites dans un programme qu'il écrit lui-même ;

— **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.

On prend comme **exemple**, pour illustrer cette génération de code, la résolution d'un système non linéaire de n équations à n inconnues pour un cas précis emprunté au domaine de l'ingénierie, à savoir **la modélisation d'une cage d'un train de laminage à chaud**.

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 (figure **11**).

Les équations qui régissent le comportement de la cage sont les suivantes :

$$f_1(F, h_2, \Phi) = h_2 - S - \frac{F + a_2(1 - e^{a_3 F})}{a_1}$$

$$f_2(F, h_2, \Phi) = F + \frac{R\xi T_1}{h_2}$$

$$-\ell k R\left(\frac{1}{2}\pi\sqrt{\frac{h_2}{R}}\arctan(\sqrt{r}) - \frac{\pi\xi}{4} - \ln\left(\frac{h_N}{h_2}\right) + \frac{1}{2}\ln\left(\frac{h_1}{h_2}\right)\right)$$

$$f_3(F, h_2, \Phi) = \arctan\left(\Phi\sqrt{\frac{R}{h_2}}\right)$$

$$-\frac{1}{2}\sqrt{\frac{h_2}{R}}\left(\frac{\pi}{4}\ln\left(\frac{h_2}{h_1}\right) + \sqrt{\frac{R}{h_2}}\arctan(\sqrt{r}) - \frac{T_1}{k\ell h_1} + \frac{T_2}{k\ell h_2}\right)$$

avec $r = \dfrac{h_1 - h_2}{h_2}$,

$\xi = \sqrt{\dfrac{h_1 - h_2}{R}}$,

$h_N = h_2 + R\Phi^2$.

Les inconnues sont la force de laminage F, l'épaisseur de sortie h_2 et l'angle neutre Φ (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'entre 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.

5.2.1 Traduction d'expressions

Il est clair que le calcul de la matrice jacobienne du cas choisi est pour le moins compliqué. En revanche, il est très facile à réaliser en utilisant un système de calcul formel comme *Maple*.

Exemple : à titre de curiosité, l'élément (2,2) de la matrice jacobienne est calculé et affiché ci-après :

```
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—1*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/1/h1+T2/k/1/h2)/2]:
Fp:=jacobian(f,[F,h2,Phi]):
Fp[2,2];
```

$$-\ell k R\left(\frac{1}{4}\frac{\pi\arctan\left(\sqrt{\frac{h_1 - h_2}{h_2}}\right)}{\sqrt{\frac{h_2}{R}}R} + \frac{1}{4}\frac{\pi\sqrt{\frac{h_2}{R}}\left(-\frac{1}{h_2} - \frac{h_1 - h_2}{h_2^{\,2}}\right)}{\sqrt{\frac{h_1 - h_2}{h_2}}\left(1 + \frac{h_1 - h_2}{h_2}\right)} + \frac{1}{8}\frac{\pi}{\sqrt{\frac{h_1 - h_2}{R}}R} - \frac{\left(\frac{1}{h_2} - \frac{h_2 + R\Phi^2}{h_2^{\,2}}\right)h_2}{h_2 + R\Phi^2} - \frac{1}{2}\frac{1}{h_2}\right) - \frac{1}{2}\frac{T_1}{\sqrt{\frac{h_1 - h_2}{R}}\,h_2} - \frac{R\sqrt{\frac{h_1 - h_2}{R}}T_1}{h_2^{\,2}}$$

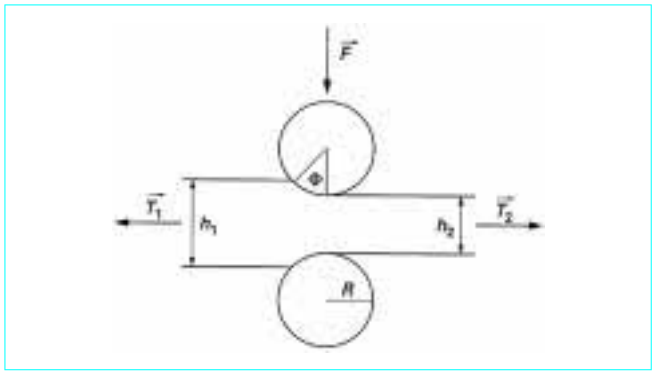

Figure 11 – Cage de laminage

Mais 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.

Nous allons transformer la matrice jacobienne obtenue ci-avant en syntaxe Fortran. Tous les systèmes de calcul formel disposent d'une fonction à cette fin. En *Maple*, la commande `fortran` réalise cette transformation. Si son argument est un tableau, tous les éléments sont transformés. Mais, les systèmes de calcul formel ont généralement des possibilités d'optimisation pour éviter de recalculer les sous-expressions communes aux éléments de la matrice, comme $\sqrt{h_2 / R}$ ou $h_2 + R\,\Phi^2$ dans le jacobien de l'exemple précédent. Cela introduit alors un certain nombre de variables auxiliaires qui contiennent ces expressions communes.

Exemple : en *Maple* cela donne (encadré 1) :

Encadré 1

```
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.

On peut bien sûr réaliser la traduction en langage C à la place du langage Fortran.

5.2.2 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 (initialisations, 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.

Exemple : en *Maple*, une boucle `do` de Fortran est produite par :

```
l print('   do 1,i=1,n'),fortran([z=z+f(i)^2]),
l print('1  continue');
```

ce qui donne le bout de code 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 permettent d'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. La *share library* est une bibliothèque de programmes *Maple* qui sont des contributions des utilisateurs et sont fournis avec la distribution standard. Il faut faire appel à des commandes spéciales pour utiliser ces fonctionnalités (voir l'exemple ci-après).

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.

Exemple : pour obtenir un sous-programme Fortran qui calcule la **matrice jacobienne de la cage du laminoir** (§ 5.2.1), avec comme arguments les variables F, h_2, Φ, T_1 et T_2, et un programme principal qui l'appelle, *Macrofort* s'emploie ainsi (encadré 2).

On peut faire la même chose en langage C, les instructions à exécuter sont différentes.

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.

Des systèmes de calcul formel comme *Mathematica* ont aussi la possibilité de lier et d'exécuter des programmes C ou Fortran.

Une autre façon de résoudre ce problème de lien, entre calcul numérique et calcul formel, est d'appeler à partir du système de calcul formel des bibliothèques numériques existantes. C'est le cas 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 numérique NAG sans sortir de *Reduce*.

Encadré 2

```
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]]]]]:
sp:=[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(sp);
```

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
      do1000, 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
```

5.3 Systèmes de CAO en automatique

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 Fortan 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 permet aussi le tracé de courbes. Un tel système dispose généralement d'un interpréteur permettant d'écrire des programmes (appelés **macros**). Le plus connu de ces systèmes est le logiciel *Matlab*. Citons aussi le logiciel français *Basile* et le logiciel *Scilab* réalisé à l'INRIA.

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 et de macros) ajoutée 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.

Pour illustrer cette dernière approche, nous allons décrire 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, programmes sources compris. De plus, il est possible d'ajouter facilement des programmes C ou Fortran à *Scilab*.

L'application présentée concerne 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 au vélo de tourner d'un angle fixé le plus rapidement possible.

La technique que nous allons utiliser fait appel au formalisme de Lagrange pour représenter un système mécanique composé de corps articulés. Il est décrit ci-après.

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L(\mathbf{q},\dot{\mathbf{q}})}{\partial \dot{\mathbf{q}}} - \frac{\partial L(\mathbf{q},\dot{\mathbf{q}})}{\partial \mathbf{q}} + \left(\frac{\partial \mathbf{f}(\mathbf{q})}{\partial \mathbf{q}}\right)^T \lambda + \mathbf{g}^T(\mathbf{q})\gamma = \mathbf{k}(\mathbf{q},\mathbf{u})$$

$$\mathbf{f}(\mathbf{q}) = 0$$

$$\mathbf{g}(\mathbf{q})\dot{\mathbf{q}} = 0$$

avec **q** vecteur des coordonnées généralisées,

$L(\mathbf{q},\dot{\mathbf{q}})$ lagrangien = énergie cinétique – énergie potentielle,

f(q) vecteur des équations des contraintes holonomes,

$\mathbf{g}(\mathbf{q})\dot{\mathbf{q}}$ vecteur des équations des contraintes non holonomes,

k(q,u) vecteur des forces et des moments appliqués,

λ, γ multiplicateurs de Lagrange associés aux contraintes (forces internes généralisées).

Ces équations donnent le système algébrico-différentiel suivant :

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} = \mathbf{F}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{k}(\mathbf{q},\mathbf{u}) - \mathbf{f}_q^T(\mathbf{q})\lambda - \mathbf{g}^T(\mathbf{q})\gamma$$

$$\mathbf{f}(\mathbf{q}) = 0$$

$$\mathbf{g}(\mathbf{q})\dot{\mathbf{q}} = 0$$

qui peut s'écrire sous la forme d'un système qu'il faut résoudre pour simuler le système mécanique :

$$\begin{bmatrix} \mathbf{M} & \mathbf{f}_q^T & \mathbf{g}^T \\ \mathbf{f}_q & 0 & 0 \\ \mathbf{g} & 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}} \\ \lambda \\ \gamma \end{bmatrix} = \begin{bmatrix} \mathbf{F} + \mathbf{k} \\ -\dot{\mathbf{f}}_q \\ -\dot{\mathbf{g}}\dot{\mathbf{q}} \end{bmatrix}$$

où **M(q)** est la matrice d'inertie généralisée.

On voit donc qu'il faut effectuer le calcul de **M** et de **F** à partir des données du lagrangien. Ces calculs matriciels sont très rapidement considérables, même si le système mécanique ne comporte que quelques corps.

Dans notre application, cela nécessite le calcul d'une matrice 23×23, 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 (§ 5.2.2)) puis ils sont compilés et inclus dans *Scilab* sous la forme de macros. Ils peuvent alors être efficacement utilisés par les fonctions de *Scilab*.

Exemple : à titre de curiosité, nous donnons ci-dessus le début de la valeur d'un élément de matrice du problème de commande, montrant ainsi la quasi-impossibilité d'une écriture à la main (encadré 3) :

Encadré 3

```
±mat(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)*1
+lambda(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)
+(...)
```

Le résultat de la simulation ou de la commande est une animation graphique du comportement du vélo. La figure **12** donne la trajectoire optimale d'un vélo pour tourner de 15 degrés (noter le contre-braquage 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.

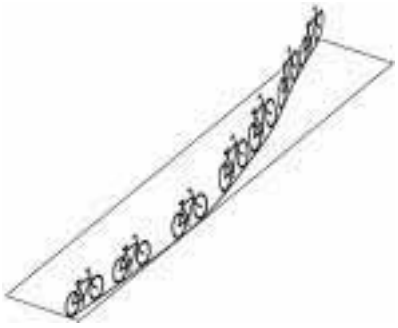

Figure 12 – Comment faire tourner un vélo de 15 degrés le plus rapidement possible

6. Autres domaines

Il est bien clair que nous n'avons pas passé en revue dans ce chapitre tous les domaines que le calcul formel peut aborder, mais nous avons décrit les principaux. On peut citer encore les domaines suivants.

En **arithmétique** et en **analyse combinatoire** le calcul formel excelle grâce à sa manipulation de vrais nombres entiers ; il est possible de résoudre des équations entières et des problèmes de dénombrement.

L'étude des **suites réelles** permet de résoudre des problèmes de récurrence (calcul des valeurs d'une suite) et de calculer les limites d'une suite. En particulier, le calcul formel donne les moyens de faire des calculs de sommes (Σ) et de produits (Π).

Les **séries** sont un outil puissant pour les problèmes qui n'admettent pas de solution exacte : on peut en effet calculer le développement en série de suites, d'intégrales ou de solutions d'équations (ordinaires ou différentielles). On peut aussi calculer le comportement asymptotique de ces mêmes objets mathématiques à l'aide des développements asymptotiques. Le calcul formel fournit des outils pour cela, mais il faut bien vérifier la justesse de la solution par la justification des opérations formelles que le système peut réaliser.

Les outils du calcul formel comme le **calcul intégral** ou les **séries génératrices** peuvent parfois être appliqués avec succès au domaine des probabilités et des statistiques ; l'**algèbre linéaire** aussi peut être utilisée.

Enfin, il faut citer l'utilisation du calcul formel pour des domaines particuliers à certaines branches de la physique comme le calcul tensoriel, la relativité, etc.