计算机代数 线性代数方程组 算法

综合报告

计算机代数介绍（下）

A. M. Cohen J. H. Davenport A. J. P. Heck

6 两个实例

本节中的两个例子选自文献中已解决的问题. 它们说明在解决这两个问题的某些步骤中，计算机代数很有用.

参加 SCAFI 项目的荷兰一家公司的研究人员对用计算机代数研究连杆机构运动学很感兴趣. 在下面的示例中我们研究四连杆问题.

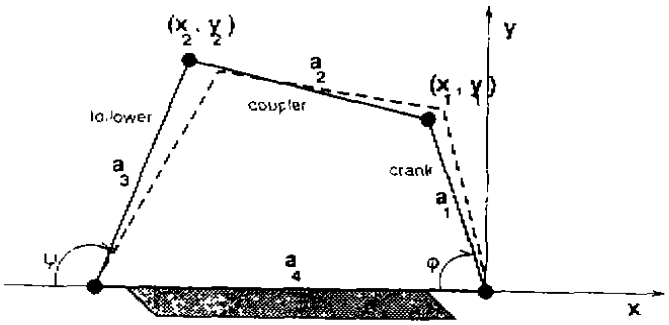

图 4 四连杆机构

我们要求解杆长，使得从动杆的角度 Ψ 满足给定的角度，角速度和角加速度. 根据方程：

$$(x_1, y_1) = (-a_1\cos\phi, a_1\sin\phi), \quad (x_2, y_2) = (-a_4 - a_3\cos\psi, a_3\sin\psi),$$
$$(x_2 - x_1)^2 + (y_2 - y_1)^2 = a_2^2$$

以及著名的三角恒等式：

$$\cos^2\phi + \sin^2\phi = 1, \quad \cos^2\psi + \sin^2\psi = 1,$$

我们来求 $\psi, \phi, a_1, a_2, a_3, a_4$ 之间的关系.

原题： An Overview of Computr Algebra. 译自： *Computer Algebra for Industry*, Vol. 16, No. 1, 1994, pp. 1–25.

通过这些方程来化简：

$$(x_2 - x_1)^2 + (y_2 - y_1)^2 = a_2^2$$

得到

```
> siderels := { x1=-a1*cos(phi), y1=a1*sin(phi), x2=-a4-a3*cos(psi),
                y2=a3*sin(psi),
>             cos(phi)^2+sin(phi)^2=1, cos(psi)^2+sin(psi)^2=1 }:
> vars := [x1, x2, y1, y2, a1, a2, a3, a4,
>          cos(phi), sin(phi), cos(psi), sin(psi)]:
> simplify( (x2-x1)^2 + (y2-y1)^2 = a2^2, siderels, vars );
```

$$a1^2 - 2\,a3\cos(psi)\,a1\cos(phi) - 2\,a3\sin(psi)\,a1\sin(phi) - 2\,a4\,a1\cos(phi) + a3^2 + 2\,a4\,a3\cos(psi) + a4^2 = a2^2$$

然后，再用下列化简指令：

```
> combine(", 'trig'): readlib(isolate)(", cos(phi-psi)):
> collect(", {cos(phi), cos(psi)}):
```

我们导出方程：

$$\cos(\phi - w) = -K_1\cos\phi + K_2\cos\psi + K_3,$$

其中

$$K_1 = a_4/a_3, \quad K_2 = a_4/a_1, \quad K_3 = \frac{(a_1^2 + a_3^2 + a_4^2 - a_2^2)}{2a_1a_3}.$$

Maple 可以从上述方程中找到 ψ 的封闭解. 这两个解对应着四连杆机构两种不同的封闭连接：

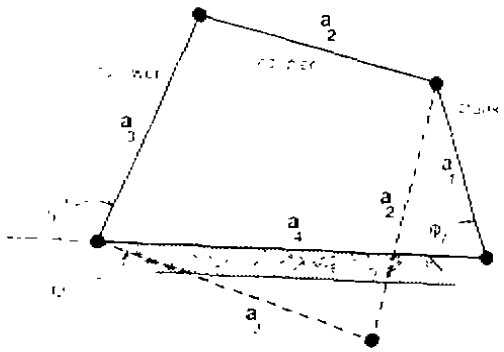

图 5　ψ^- 和 ψ^+ 对应的两个解

解答 ψ^+ 可以按下述方式化简：

```
> solve( cos(phi-psi) = -K1*cos(phi) + K2*cos(psi) + K3, psi ):
> psiplus := subs( sin(phi)^2=1-cos(phi)^2, "[1] );
```

$$psiplus := 2\arctan(1/2\,(\ 2\sin(phi)$$

```
+ (4  8 cos(phi) K2  4 K1^2 cos(phi)^2
+ 8 K1 cos(phi) K3 + 4 K2^2 - 4 K3^2)^1/2
)/(- cos(phi) + K1 cos(phi) + K2 - K3))
```

关于角速度和角加速度的关系也可以用相同的方式很容易地导出. 首先, 我们定义 ϕ 和 ψ 为时间 t 的函数, 并引入 ϕ 和 ψ 的一阶和二阶导数的别名:

```
> alias( phi=phi(t), psi=psi(t),
>        w_phi=diff(phi(t), t),   w_psi=diff(psi(t), t),
>        a_phi=diff(phi(t), t$2),   a_phi=diff(psi(t), t$2) )
```

对含有 ϕ 和 ψ 的方程求导, 我们推出下列关系:

```
> eqn1 := cos(phi-psi) = -K1*cos(phi) + K2*cos(psi) + K3.
> eqn2 := diff( eqn1, t):   eqn3 := deff( eqn1, t$2):
> eqns := {eqn. (1..3)};
eqns := {
- sin(- psi + phi) (- w_psi + w_phi) = K1 sin(phi) w_phi
- K2 sin(psi) w_psi,
cos( - psi + phi) = - K1 cos(phi) + K2 cos(psi) + K3.
- cos(- psi + phi) (- w_psi + w_phi)^2
- sin(- psi + phi) (- (a + psi) + a_phi)
= K1 cos(phi) w_phi^2 + K1 sin(phi) a_phi - K2 cos(psi) w_psi^2
  K2 sin(psi) (a + psi)
      }
```

对于给定的角度, 角速度和角加速度, 这一方程组是关于 K_1, K_2 和 K_3 的线性方程组. 其解答可以利用命令 "solve" 求出:

```
> sol := solve( eqns, {K1, K2, K3 } ):
```

一旦求出 K_1, K_2 和 K_3 的值, 我们就可以求出连杆长度 a_1, a_2, a_3, 和 a_4.

```
> sys := { K1= a4/a3, K2=a4/a1, K3=(a1^2+a3^2+ a4^2-a2^2)/(2*a1*a3) }:
> solve( sys, {a1,a2,a3} ):
> solution := "[2];
 solution :=
{a1 = a4/K2, a2 = a4(-2 K3 K1 K2+K1^2+K2^2+K1^2K2^2)^1/2 / K1 K2, a3 = a4/K1}
```

于是, 当给定了主动杆的角度, 角速度和角加速度, 从动杆也将具有确定的角度, 角速度和角加速度. 而上述公式给出连杆机构的杆的长度.

我们取 [HD] 中问题 10-6 中给出的数值

```
> values := { phi=Pi/2, w_phi=3, a_phi=0, psi=Pi/2, w_psi=6/5,
> a_psi=81/50 }:
> eval(subs( values, sol ));
                    {K3 = 1, K2 = 2, K1 = 4/5 }
```

```
> assign(''): simplify( solution );
                        {a3 = 5/4 a4, a1= 1/2 a4, a2= 5/4 a4}
```

取基杆的长度为 4，并令 $\varphi = \frac{\pi}{3}$，我们得到四连杆机构的图形如下：

```
> a4 := 4:    assign(" "):
> alias( phi=phi ):    phi := Pi/3:
> x1 := evalf( -a1*cos(phi) ):      y1 := evalf( a1*sin(phi) ):
> x2 := evalf( -a4-a3*cos(psiplus) ):    y2 := evalf( a3*sin(psiplus) ):
> plot( [0,0,x1,y1,x2,y2,-a4,0], style=LINE );
```

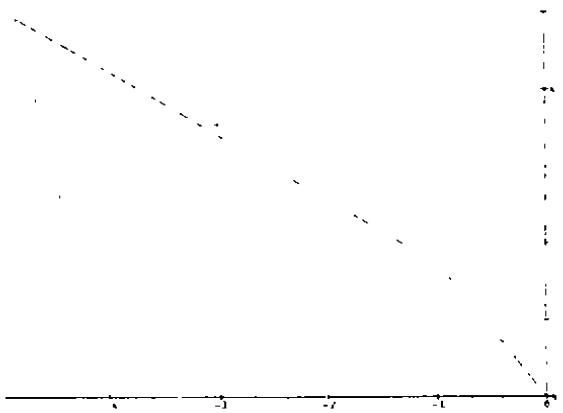

图 6　$\varphi = \pi/3$ 时的连杆机构

因为 Maple 可以把从动杆的角度用连杆长度和主动杆的角度来表示，所以不需要更多的工作就可以作出角度之间依赖关系的图形：

```
> phi := evaln( phi ):
> plot( psiplus, phi=0..2*Pi );
```

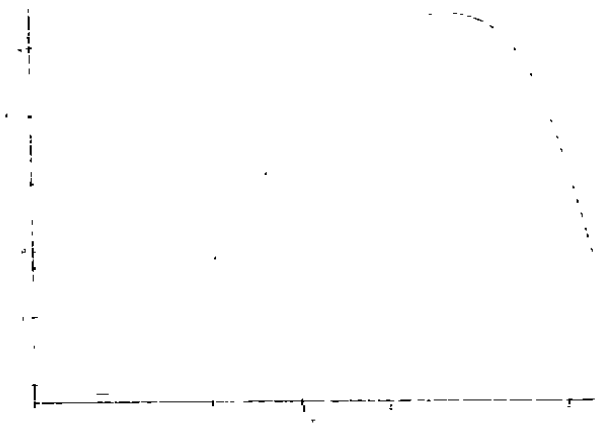

图 7　角度之间的关系

连杆机构运动学的动画片也许比图形更加生动. 这可以通过 Maple 或 Mathematica 的 Window 版本作出.

6.2 热传导

现在我们考虑隔热层传热的模型. 这个模型将帮助我们分析隔热墙一侧在极高温度下的热传导现象. 考虑以 x 为坐标的一维空间, 它垂直于隔热层, 假定该空间是均匀的, 无穷长的.

表示这个模型的微分方程是:

$$\frac{\partial T}{\partial t}=\frac{\partial}{\partial x}(k(T)\frac{\partial T}{\partial x}),$$

其中, $T=T(t,x)$ 代表热量, 它是时间 t 和空间坐标 x 的函数, 函数 $k(T)$ 是隔热层材料的热传导系数.

对于普通材料而言, $k(T)$ 可以取为常数. 这时我们可以用分离变量法来解上述微分方程 (见 [Witt]). 对于特殊的材料. 例如: Prosial 和 Norcoat, $k(T)$ 不再是常数. 为了分析这些情形, 我们假定 $k(T)=T^a$, 从而仍然可以应用分离变量法. 令 $T(t,x)=f(t)g(x)$, 代入微分方程, 然后分离变量, 我们得到

$$\frac{f_t}{f^{a+1}}=g^{a-2}(ag_x^2+gg_{xx}),$$

于是方程的两侧必然是常数. 假定该常数是 A, 方程的左侧很容易解. 如利用 Maple 的指令 **dsolve(lode, f(t), 'explicit')**, 其中 lode 代表 diff(f(t), t) = A*f(t)**(a+1), 由此得到:

$$f(t)=(\frac{-1}{atA-C_1aA})^{\frac{1}{a}}.$$

解右侧的方程

$$ag_x^2+gg_{xx}=Ag^{2-a}$$

比较困难. 如果 $a=0$. 则软件包可以轻松地发现解:

$$C_1e^{\sqrt{A}x}+C_2e^{-\sqrt{A}x}.$$

如果 $a\neq 0$, Maple 输出

$$x=\int_0^{g(x)}\frac{\sqrt{a+2}y^a}{\sqrt{2Ay^{a+2}+(a+2)C_1}}dy-C_2.$$

当 $a=1$ 时, 该解为

$$x=\int_0^{g(x)}\frac{\sqrt{3}y}{\sqrt{2Ay^3+3C_1}}dy-C_2.$$

正如 §3.2 所示, 这个结果意味着原微分方程不存在封闭形式的解 (除非参数取某些特殊的值. 例如 $a=1, C_1=C_2=0$ 时 $g(x)=Ax^2/6$). 进一步的分析也许可以通过迭

代法，或特征空间分解法来进行．计算机代数软件包在进行这类分析时也可能很有用．

另一个方法是用 Maple 计算该微分方程在 $x=0$ 处的泰勒数解．指令：**dsolve({rode, g(0)=C0, D(g)(0)=C1 }, g(x), 'series')** 的输出结果为：

$$g(x) \cong C_0 + C_1 x + \frac{AC_0^{2-a} - aC_1^2}{2C_0}x^2 + \frac{C_1(A(1-3A)C_0^{2-a} + a(2a+1)C_1^3)}{6C_0^2}x^3,$$

其中 rode 是 a*diff(g(x), x)**2 + g(x)*diff(g(x), x$2) = A*g(x)**(2-a)．

上述两个例子展示了现代计算机代数所能做到的一些事情．讨论更多，更大规模的应用正是本书的目的．

7 算法

我们将讨论计算机代数的较为理论的一面，一个算法在计算机代数系统中实现之前，它必须经过以下两个阶段：即工业生产中的研制和开发阶段．很多成功的算法诞生于较为理论的数学层次上．如果确信某个算法将有较好的实际效果，则我们把它写成软件，并且对该软件进行检验和调试，正如开发工业产品时所做的一样．在本节中，我们重点讨论算法的研究阶段．

7.1 算法的定义

请注意以下两句话有很大区别：

1 我的程序可以解决某些问题．

2 我的程序可以解决属于某个问题类的所有问题．

第一句话反映了人工智能研究者的态度；而第二句话却属于算法的范畴．第二句话中所提及的程序必须实现一个算法，它必须对一类给定问题的每一个特例都给出正确的答案．

对于较为容易的问题类，这样的要求是显然的．如果你要一个袖珍计算器求两个整数的积，而计算器回答："对不起，我想不出该怎样做．"你当然不会对这个计算器有好印象．然而，对于比较复杂的问题，算法也许是根本不存在的．例如：求解丢番都方程 (即求多项式方程组的整数解)．

也许很难准确地定义什么是给定的问题类．一个经典的例子是积分．我们常听说 "e^{-x^2} 没有积分"．但是这句话的含义到底是什么呢？从某种意义上讲，这句话显然是不对的？因为 e^{-x^2} 是连续函数，所以是可积的．而且它的数值积分表自从高斯时代就已经存在了．或许这句话的意思是 e^{-x^2} 的积分无法用公式来表达．它的意思是：虽然我们知道

$$\int \frac{1}{x}dx = \log x, \tag{1}$$

但并不存在函数 $f(x)$，使得：

$$\int e^{-x^2}dx = f(x). \tag{2}$$

另一方面，我们可以十分肯定地写下：

$$\int e^{-x^2}dx = \frac{\sqrt{\pi}}{2}\mathrm{erf}(x). \tag{3}$$

也许苛刻的人会认为 (3) 实际上是函数 erf 的定义，但也许可以同样地说 (1) 是 log 的定义.

为了理解含糊的命题 "没有积分" 的真正含义，我们不得不首先定义什么是初等函数. 为了简单起见，假定所有函数都是 x 的函数. 对于变量 $x, y, \ldots$，我们用 $K(x, y, \ldots)$ 记关于这些变量的有理函数域. 初等函数域是域 $K(x, \theta_1, \ldots, \theta_n)$，其中 θ_i 是

以 $K(x, \theta_1, \ldots, \theta_{i-1})$ 中元素为系数的某个多项式的根，或者

$K(x, \theta_1, \ldots, \theta_{i-1})$ 中某个元素 η 的对数 (即满足方程 $\theta_i' = \eta'/\eta$)，或者

$K(x, \theta_1, \ldots, \theta_{i-1})$ 中某个元素 η 的指数函数 (即满足方程 $\theta_i' = \eta'\theta_i$).

一个初等函数就是初等函数域中的一个元素，而命题 "e^{-x^2} 没有积分" 意思是不存在初等函数满足 (2).

于是，有多少可作为被积函数以及积分的函数类，就有多少关于积分算法的不同定义. 最近 Brostein 解决了下列积分问题 (见 [Br]): 假定被积函数为初等函数，设计一个积分算法使得当被积函数的积分是初等函数时，计算该积分；否则，证明该被积函数的积分不是初等函数.

然而，设计算法和实现算法是两件不同的事. 很多比较复杂的计算机代数算法完全没有实现. 另一部分算法仅仅实现它们的特殊情形. 这一现象在计算机代数中比在数值分析中更为普遍. 其原因是一个算法往往依赖于另一个算法和很多较为复杂的数学知识. 例如：上面提到的 Brostein 积分算法需要调用以下算法：

Risch 结构定理的一个有效表示，用它来判定初等函数间的依赖关系.

判定代数曲线上的一个除子是有挠的还是无挠的 (如果是有挠的，挠率是什么)

判定代数曲线 (也许是在多维空间里) 是否绝对不可约.

虽然上述算法甚至它们的实现都存在，但这些软件：

在不同的计算机代数系统；

有局限性 (例如，判定绝对不可约性算法只对平面代数曲线适用)；

需要不同的数据结构；

是 "实验性的" 而不是 "生产力".

类似的经验也可以在其它较为先进的算法中找到. 为了实现一个先进的算法，实现算法的工作人员需要对基本算法进行大量的工作，其程度超过一般人的想象.

下一节中，我们将简短地描述一类有决定意义的算法.

7.2 Gröbner 基

我们已经知道计算机代数系统可以解适度规模的多项式方程组. 算法所依赖的策略是现代计算机代数的经典工作. 关于这一题目的两篇经典文章是 [Be] 和 [Mc].

Gröbner 基理论综合了出自两种特殊情形的主要原理. 第一个特殊情形是求解方程组

$$f_1 = f_2 = \cdots = f_N = 0,$$

其中 $f_1, \ldots, f_N$ 是关于 x 的一元多项式. 在这种情况下我们可以通过欧基里算法运用次数约化原理. 首先, 我们计算 $f := \gcd(f_1, f_2)$, 即前两个多项式的最大公因子. 回忆最大公因子是通过把多项式对 f_1, f_2 约化为另一个多项式对. 约化的办法是从次数较高的多项式减去次数较低的多项式从而得到一对新的多项式, 其中出现的最高次数降低了. 重复这一过程直到多项式对中的某个成员为零, 则该对中的另一个多项式即为所求的最大公因子. 然后我们计算 f 和 f_3 的最大公因子, 等等. 在最后一步, 我们计算 $f := \gcd(f, f_N)$, 从而得到 $f = \gcd(f_1, f_2, \ldots, f_N)$. 于是, f 的任何解都是整个系统的解, 而且反过来也成立, 即整个系统的解也是 f 的解. 这样, 我们就把求解方程组的问题化为求解一个多项式的问题.

另一个特殊情形是求解线性方程组. 现在, 变元的个数是任意有限个, 但每一个方程必须是齐 1 次的. 我们知道高斯消去法可以用来解线性方程组. 其基本原理是消去变元 (将一个线性多项式乘以一个适当的数与另一个线性多项式相减) 如果这一过程终止, 我们得到如下的方程组

$$(U_m, M)\begin{pmatrix} x \\ y \end{pmatrix} = 0,$$

其中, $x = (x_1, \ldots, x_m), y = (y_1, \ldots, y_k)$ 是变元 (对变元进行适当排序), U_m 是 m 阶上三角形矩阵, 它的对角线上元素为 1. M 是 $m \times k$ 阶矩阵. 现在 y 是自由变元组, 而变元 x 可由变元 y 线性表示 (即 $x = -U_m^{-1}My$). 从计算角度而言每一个 $x_j, j = m, \cdots, 1$, 可以通过逐步回代法写成 y 的线性组合.

现在, 我们考虑一个一般的多项式系统

$$f_1(x) = f_2(x) = \cdots = f_N(x) = 0,$$

其中 $x = (x_1, x_2, \ldots, x_m)$. 解这个系统的方法是欧基里德方法和高斯消去法的结合. 它从所有单项式 $x_1^{a_1} \cdots x_m^{a_m}$ $(a_i \in \boldsymbol{Z})$ 的全序开始, 要求该序与单项式的乘法相容, 即若 $m_1 < m_2$, 则对于任意的单项式 $m_3, m_3 m_1 < m_3 m_2$ 成立, 而且 $1 = x_1^0 \cdots x_m^0$ 是最小的单项式. 算法的正确性与满足上述两个条件的序的选择无关. 但是序的选择会影响算法的效率. 为了简单起见, 我们只需要记住基于变量排序的字典序是十分有效的.

令 G 是一个多项式的集合. 我们认为 G 表示了一组多项式方程组 $g = 0, \forall g \in G$. 任取 G 中的两个元素 f 和 g, f 和 g 的 S-多项式 定义为:

$$S(f,g) = \begin{cases} 0 & \text{如果} f = 0 \text{或} g = 0 \\ \dfrac{\mathrm{lt}_g}{\gcd(\mathrm{lm}_f, \mathrm{lm}_g)} f - \dfrac{\mathrm{lt}_f}{\gcd(\mathrm{lm}_f, \mathrm{lm}_g)} g & \text{如果} fg \neq 0 \end{cases}$$

其中 lm_f 代表 f 中对于给定序的最高单项式，lt_f 代表 f 中最高单项式 lm_f 所在的项 (即该单项式与它的系数的乘积).

重要的是 f 和 g 的任何零点也是 $S(f,g)$ 的零点.

让我们现在来考虑在上面两个特殊情形下，S- 多项式的含义. 在一元情形下，S- 多项式恰好是欧基里得算法的第一步：从高次多项式中减去低次项式的适当倍式 (主变元单项式的一个纯量倍). 从而降低高次多项式的次数. 对于线性情形，如果两个多项式 f 和 g 同时含有最高变元，则 S- 多项式是 f 和 g 的一个线性组合. 但若 f 和 g 所含的最高变元不相同，我们将得到一个二次多项式. 人们不希望在线性情形产生二次多项式. 为了把高斯消去法放入 Gröbner 基理论的框架，我们应该认识到在这种情形下是有缺陷的，它应当避免产生二次多项式.

我们来计算 $f = x^2y - 1$ 和 $g = xy^2 - 1$ 的 S- 多项式. 注意到 $\mathrm{lm}_f = x^2y$ 和 $\mathrm{lm}_g = xy^2$. 于是，无论如何对单项式排序，都有 $S(f,g) = yf - xg = y - x$ 但是为了确定多项式 $y - x$ 的首项，我们必须定义 $x < y$ 或者 $y < x$. 假定我们选择 $x > y$, 则 f 和 $y - x$ 的 S- 多项式为 $-f - xy(y - x) = -g$(这并不给出新的信息), 但 g 和 $y - x$ 的 S- 多项式是 $-g - y^2(y - 2) = 1 - y^3$. 这是一个一元多项式. 这说明方程组 $f(x,y) = g(x,y) = 0$ 的任何零点 (x,y) 都满足方程 $y^3 - 1 = 0$. 另一方面，对给定的 y, 方程 $y = x$ 确定了 x. 于是，我们找到了原方程组的三组解：$x = y = e^{2\pi ki/3} (k = 0, 1, 2)$. S- 多项式的计算可以导出方程组的解不是偶然的.

Gröbner 基的算法运行如下：假定给定了一个多项式方程组 G 的字典序. 从 G 中选取 f 和 g, 并计算它们的 S- 多项式 c. 然后检验 c 是否含有 "新" 的信息. 检验过程是从 c 减去 G 中尽可能多的多项式的倍式，从而降低 c 中首项的序. 其结果称为 c 关于 G 的约化形式. 在上面的例子中，$-g$ 不含有新的信息是因为 G 中的元素 g 乘以 -1 与 $-g$ 相减等于零. 一般来说，"c 不含新的信息" 一语的确切的说法是，它关于 G 的约化形式是零. 于是，如果 g 的约化形式不为零，则把该约化式加到 G 中 (因为它带来了新的信息). 然后再对新的 G, 选择一对多项式，重复上述过程.

当 G 中没有一对多项式会给出任何新信息时，算法终止. 所以 G 是不断扩大的. (当然，在比较老练的版本中，G 中的某些元素被不断地删去.) 该算法的终止性不是显然的. 然而，数学理论 (特别是 Hilbert 基定理) 保证了算法的终止性. 称算法输出系统为 S, 它是一个 *Gröbner* 基, 这意味着 S 中的任意一对多项式的 S- 多项式的约化形式为零. 让我们再一次考虑前面两种特殊情形. 在一元多项式的情形，从 G 出发得到的含有 G 中所有元素的最大公因子的集合 S 将是一个 Gröbner 基. 而在线性情形，在高斯消去法设定的变元序下，该消去法得到的行向量所对应的线性多项式，组成一个 Gröbner 基. 上面所讲的算法称为 *Buchberger* 算法.

现在我们再来考虑求解方程组 G 的方法. 假定 y 是关于给定的单项式序的最小变元，则系统 S 至多含有一个关于 y 的一元多项式 S_1(忽略所有 S_1 的倍数), 则 G 的任何一个解都是 S_1 的解. 在上面的例子中 $G = \{f, g\}$, 其中 $f = x^2y - 1, g = xy^2 - 1$,

Buchberger 算法给出了 Gröbner 基

$$S = \{f, g, x - y, y^3 - 1\}$$

(注意 $\{x-y, y^3-1\}$ 也是 G 的一组 Gröbner 基!), S 中仅含有 y 的多项式是 $S_1 = y^3 - 1$. 由此出发，上三角形构作轮到其它变元. 如果 x 是依给定单项式序的下一个变元，为了求 x 的值，我们只需要把 y 的值代入到含有 x 和 y 的多项式中，就可以得到一组仅含有 x 的多项式，再利用欧基里德算法求出 x 的值，然后再继续这一回代过程. 在示例中，这个回代过程是将 $y = e^{2\pi ki/3}$ 代入 $x - y = 0$.

我们再举一例，方程组来自 §2:

```
> eqns := {x^2+y^2=l^2, x^2+z^2=m^2, x=a+b, h*x=a*z, h*x=b*y}:
```

定义这组方程的多项式是:

```
> polys := map( lhs-rhs, eqns );
polys := {x - a - b, h x - a z, h x - b y, x^2 + z^2 - m^2, x^2 + y^2 - l^2}
```

我们采用一个将 Buchberger 算法和多元多项式因式分解相结合的方法来解这组方程(见 [Cz] 和 [Dan]. 指令为

```
> sys := gsolve( polys, {l,m,h}, {x,y,z,a,b} )
```

它给出 sys 的值:

$$
\begin{aligned}
&[[a, b, y^2 - l^2, z^2 - m^2, x],\\
&[x^5 + (2l^2h^2 - 2h^2m^2)a + (-h^2m^2 - 3l^2h^2 + l^2m^2)x + (4h^2 - m^2 - l^2)x^3,\\
&(2l^2h^2 - 2h^2m^2)b - x^5 + (l^2h^2 - l^2m^2 + 3h^2m^2)x + (m^2 - 4h^2 + l^2)x^3,\\
&(-2l^2 - m^2 + 4h^2)x^4 + x^6 + (2h^3l^2 - 2h^3m^2)y + l^4h^2 - l^4m^2 + 3l^2h^2m^2\\
&\quad + (l^4 - 5l^2h^2 - 3h^2m^2 + 2l^2m^2)x^2,\\
&(2m^2 - 4h^2 + l^2)x^4 + (2h^3l^2 - 2h^3m^2)z - x^6 - h^2m^4 - 3l^2h^2m^2 + l^2m^4\\
&\quad + (5h^2m^2 + 3l^2h^2 - 2l^2m^2 - m^4)x^2,\\
&(4l^2m^2 - 6l^2h^2 - 6h^2m^2 + m^4 + l^4)x^4 + (-2l^2 - 2m^2 + 4h^2)x^6 + x^8\\
&\quad + l^4m^4 - 2h^4m^2l^2 + h^4m^4 - 2l^4h^2m^2 - 2l^2h^2m^4 + l^4m^4\\
&\quad + (8l^2h^2m^2 - 2l^2m^4 + 2h^2m^4 + 2l^4h^2 - 2l^4m^2)x^2]].
\end{aligned}
$$

在表中的每一个子系统的解都是原方程组的解，但每一个子系统的变元都尽可能地被相继消去和分离了. 每一个子系统都是对纯字典序 $a > b > z > y > x$ 的约化 Gröbner 基.

让我们来考察一下第二个子系统. 在最后一个 (第 5 个) 多项式中， a, b, z, y 都不出现. 所选取的单项式序保证了任何一个由系统产生的、不含 a, b, z, y 的多项式都是这个多项式的倍数. 特别地，我们可以解这个多项式关于 x 的根. 然后将其值代入到第 4 个方程中，从而第 4 个方程变为关于 z 的一元多项式，然后再如法炮制地解整个方程组.

与 §2 中所述相比较，第二个子系统的第 5 个方程的次数是 8 而不是 4, 这一点也许令人费解，为了理解为什么在 §2 中 Maple 得到了一个 4 次多项式，我们作变量代换 $x = \sqrt{-Z^2 + l^2}$:

```
> sys[2][5];
x^8 + (- 2 l^2 + 4 h^2 - 2 m^2) x^6
 + (- 6 m^2 h^2 - 6 h^2 l^2 + m^4 + l^4 + 4 m^2 l^2) x^4
 + (8 m^2 h^2 l^2 + 2 h^2 l^4 + 2 m^4 l^2 - 2 m^4 l^2 - 2 m^2 l^4) x^2 + h^4 l^4
 + m^4 h^4 - 2 m^2 h^4 l^2 - 2 m^4 h^2 l^2 - 2 m^2 h^2 l^4 + m^4 l^4
> factor( subs( x=sqrt(-Z^2+l^2), " ) ):
> map(collect,",Z);
(Z^4 + 2 Z^3 h - (l^2 - m^2) Z^2 - (2 l^2 h - 2 h m^2) Z - l^2 h^2 + h^2 m^2)
(Z^4 - 2 Z^3 h - (l^2 - m^2) Z^2 - (- 2 l^2 h + 2 h m^2) Z - l^2 h^2 + h^2 m^2)
```

注意到第二个因子与 §2 中所得到的多项式相同，而第一个因子可通过变换 $Z \to -Z$ 转变为第二个因子.

7.3 可解性

我们对算法的定义是非常乐观的. 对于某些问题类，算法解还没有找到；而对另一些问题类已知算法解根本不存在. 这些，连同效率等非理论性的理由，导致了大量的 "部分算法". 例如：已经知道不存在这样的算法，它可以对一个群的任一组给定的生成元及其关系，判定两个字是否代表同一群元素. 尽管如此，仍有很多程序试图解决这一问题的某些特殊情形. 这些程序保证：如果它给出一个答案，则这个答案是正确的.

由于实际需要，还产生了随机算法 (即在执行过程中有随机选择). 这些算法提供概率意义下的正确答案. (于是，它们不是 §7.1 中所讲的严格意义下的算法). 一个随机算法被称为 *Monte Carlo* 方法，如果该算法给出正确答案的概率随着运行时间的增加而增加. 一个 Monte Carlo 方法被称为是 *Las Vegas* 的，如果它或者输出一个正确的答案，或者告诉用户算法失败 (即输出 "don't know").

在 §4.1 中，我们计算

$$A = -7x^6 - 5x^5 - 8x^4 + 4x^3 + 3x^2 + 8x - 3$$

和

$$B = 5x^6 - 4x^5 - 2x^4 + 3x^3 - 10x^2 - 9x + 4$$

的最大公因子. A 和 B 的最大公因子整除 A 和 B, 模任何素数后也如此. 假设 A_p 为 A 模 p 的约化形式，等等. 则我们刚刚证明了 $\gcd_p$(A,B) 整除 $\gcd(A_p, B_p)$. (在模 p 的意义下). 于是，如果我们能找到一个素数 p 使得 $\gcd_p(A_p, B_p) = 1$, 则可以推出 $\gcd(A, B) = 1$. 模 2 后，这两个多项式可以化为

$$x^6 + x^5 + x^2 + 1 \text{ 和 } x^6 + x^3 + x.$$

很容易证明这两个多项式模 2 后是互素的. 事实上, 除去31 和 89053721. 其它素数都会告诉我们这两个多项式是互素的. 这个例子说明我们可以设计一个判定两个多项式是否互素的 Las Vegas 方法: 我们选择一个素数 p(它必须不整除首项系数的最大公因子), 然后判定这两个多项式模 p 后是否互素. 如果这两个多项式的确是互素的, 则除去有限多个素数 (它们整除这两个多项式的结式), 其它的素数都能指出互素性. 这些方法可以推广到计算整系数一元多项式的最大公因子. 所得到的方法一般来讲比直接算法要快得多. 它也可以推广到多元多项式和其它问题, 从而给出一批 *模算法*.

一般的 Monte Carlo 算法给我们更少的保证. 它们的输出只保证按一定概率正确. 一个典型的例子是 Rabin 的素数判定法. 如果该算法说: 整数 N 是一个合数, 我们可以确信; 但如果它说 "N 也许是一个素数", 则它实际上是说: "N 或者是素数, N 或者是合数, 是合数的概率小于 25%." 在文献 [D] 中有关于 Rabin 算法的准确性的讨论. 关于素判定的 Las Vegas 方法也是存在的.

在乐观的方面, 解决古老数学问题的新算法仍在不断地发现. 一个漂亮的例子是最近发现了解指数多项式方程

$$p(x, y, e^x)$$

的方法, 其中 $p(x, y, z)$ 是一个多项式 [Rn, Vor].

7.4 复杂性

计算机代数系统的大量应用和由于系统的局限性导致的死机现象一直促使人们去发现更为有效的算法. 算法效率的一种度量方法是用该算法软件在给定的机器上, 对一类给定的问题进行求解, 从而得到该软件所用的时间和空间的统计数据.

更为理论的, 渐近度量来自于计算机科学中的复杂性理论, 这里要给出算法计算量的界, 计算量是输入规模的函数, 它是按照执行算法所需的初等操作次数来估计的. 例如: 排序算法 Heapsort 的复杂度估计为:

Heapsort 用不多于 $3n \log n$ 次交换可以对一个 n 元数组排序.

Heapsort 的程序由操作 "isinverteds" 和 "interchages" 组成. 其中运算 *interchage*(i,j) 交换数组的第 i 项 a_i 和第 j 项 a_j, 而运算 *isinverted*(i,j) 输出 "**true**", 如果

$$((a_i < a_j) \text{ and } i > j) \text{ or } (a_i > a_j, \text{ and } i < j)),$$

否则输出 "**false**". 指令 *isinverted*(i,j) 告诉我们是否需要交换 a_i 和 a_j 以得到正确的相互位置. 用 "isinverted" 检验所有的元素对, 并在必要时用 "interchange" 作轮换, 也就是利用下列指令:

if *isinverted* (i,j) **then** *interchange* (i,j) **fi**

将导致一个复杂度为 $\binom{n}{2}$ 的算法. 于是 heapsort 比这个算法要好得多 (注意我们没有考虑 *isinverted* 用的时间). 这种比较方式对分析算法的有效性很有用. 人们通常用

初等比特 (bit) 运算，而不用 *interchange* 来表示整数，多项式，和其它类似对象的算法的复杂性. 依据这一观点，(二进制) 长度为 n 的两个数的通常乘法需用 Cn^2 次初等操作，而利用快速付里叶变换，我们可以得到 $C'n\log n\log\log n$ 的复杂度. 不幸地是：常数 C' 比 C 要大得多. 只有当 $n \geq 20{,}000$ 时，第二个算法才比第一个算法有效. Paul Zimmermann 最近在 Inria/DecPrl 的 BigNum 软件包中实现了应用快速付里叶变换的整数乘法，并比较了三种乘法算法的效率. 在 DecStation/5000 上，试验结果如下：当整数十进位数小于 600 位时，普通的乘法最快，当整数位数在 600 和 20,000 位之间时，所谓的 Karatsuba-Toom 算法最快，除此之外，付里叶变换法是最快的.

在解多项式方程方面，最近一个比较好的结果是 Faugère-Gianni-Lazard-Mora 算法 (见 [L]). 当多项式组是零维时，他们的算法可在 $O(n^3)$ 次内求出解，其中 n 是该方程组解的个数. 但数 n 与输入的方程组的规模相比，可能很大. 例如：方程组

$$\begin{aligned}
x_1+x_2+x_3+x_4+x_5 &= 0\\
x_1x_2+x_2x_3+x_3x_4+x_4x_5+x_5x_1 &= 0\\
x_1x_2x_3+x_2x_3x_4+x_3x_4x_5+x_4x_5x_1+x_5x_1x_2 &= 0\\
x_1x_2x_3x_4+x_2x_3x_4x_5+x_3x_4x_5x_1+x_4x_5x_1x_2+x_5x_1x_2x_3 &= 0\\
x_1x_2x_3x_4x_5 &= 1
\end{aligned}$$

有 105 个解.

事实上，有一系列上述类型的方程组，它的参数是变元的个数 d，上面的系统是 $d=5$ 的特例. 最近 $d=6$ 和 $d=7$ 的情形已被解决. 当 $d=8$ 时，理论证明该系统不是零维的. 如果 Buchberger 算法的复杂度关于 d 确是双指数型的，如 [MaMe] 所指出，根据 §4.5 的观点，寻找更好算法的需求就依然存在.

8 算法实现

怎样在计算机上实现算法是一个重要的问题. 事实上，因为各种特殊情形需要用不同的办法，一个算法可能有多个最佳的软件实现. 一个简单的例子是计算 Gröbner 基时怎样选择系数. 如果系数是模 17 的整数，那么我们就没有必要考虑中间表达式膨胀的问题. 但如果系数是整数，则中间表达式膨胀就会成为一个问题. 在 Maple, Mathematica 和 Reduce 等通用软件，都有较好的能力来计算比较一般和规模适当的 Gröbner 基. 更大的多项式系统则更有可能被专用软件包解出. 这样的软件包有：Macaulay, CoCoA, SAC, FORM 和 Bergman 等. 上面提到的软件包都有自己的特点和局限性. 向数学界推荐一个最好的软件包是很困难的. 欧洲联合研究项目 (POSSO) 致力于编制一个解方程器，它是解各类可计算系数域上多项式方程组的工具箱. 这个解方程器既是应用程序，又是一个独立的系统.

对于这类系统的期望一直是很高的. 与上段类似的评论也适用于项目 CATHODE, 它致力于构造一个利用计算机代数处理微分方程的工具箱. 如果该项目成功，

它将提供一个分析微分方程解的工具箱. 在数值计算和图形工具帮助下，它将提供一系列代数和符号方法及算法. (例如：封闭形式解，纯形式解，近似展开等等).

公开密钥密码学的应用极大地刺激了因子分解和素数判定的研究. 近二十年来，这方面的结果层出不穷. 一个引人注目的结果是：尽管分解一百位以上的整数仍然很困难，但我们已经可以判定一个多于一千位的整数是否是素数.

8.1 语言

当使用计算机代数软件时，系统的指令是由一种反映系统构造原理的语言所提供的. 大多数计算机代数系统所提供的编程功能都有 C 、PASCAL 、LISP, 或 ALGOL68 等语言所有性质. 但用户不用费心去考虑诸如内存管理和其它技术细节.

当考察一个新的系统时，需要谨慎地检查赋值语句是怎样解释的. 考虑下列程序 (它在非符号系统中是违法的，原因是 b 没有值):

```
a := b;
b := c;
a;
```

在某些系统中的输出是 c, 而在另一些系统输出是 b. 在前一种系统中，为了防止 a 取值 c, 我们应当利用类似于下面的指令：

```
a := copy(b);
b := c;
a;
```

其中 copy 指令把 b 的值，而不是 b 本身赋给 a. 观察这一问题的方法是注意到 a 是一个正常的程序变元，c(假定它从未被用过) 是做为一个数学变量来使用的，而 b 则是身兼二职. 至于哪种解释更为主要，不同系统可能有不同的处理. 在 Maple 中，指令序列 $a := b; b := c;$ 意味着 a 是符号 b, 而 b 的内容是 c; 于是， Maple 可以计算 $x := y + 1, y := 2x;$ 但系统会警告用户使用了递归的定义，只有当用户对 x 或 y 赋值时，才可能出现无终止式的执行 (死机).

除了 AXIOM 以外，其它重要的计算机代数系统对数学对象的解释都比较宽松. 数学对象的表示可以在 TEX, 多种程序语言，计算机代数系统和自动推理系统 (例如：Automath) 中找到. 这些表示不是唯一的，甚至很难从一种翻译为另一种. 为了理解数学对象的表达问题，我们来考虑 ab 的多意性. 它可能表达

一个单独的变量 ab, 或

a 和 b 的乘积 ab, 或

算子 a 作用于 b, 即 $a(b)$.

也许要通过大量的研究才可能发现用计算机表示数学对象的方法、特别是这些对象出现在下列不同阶段时：

通过电子媒介就某个数学问题进行交流；把计算机内的数学文章打印在纸上；验证一个代数运算或证明是否正确；操作和估价包含数学结果和定理的数据.

可以预计在不久的将来计算机代数系统和其它程序之间关于数学对象的交流将变得更加透明.

8.2 类型

类型是算法实现的另一个重要方面，为了描述表示对象的准确含义，表达式的取值范围显得十分重要. 例如：假定用户输入 $\frac{x-1}{x^2-1}$，如果 x 还没有用过，则该表达式的含义是一个有理函数. 但如果 x 已经被赋值，例如 x 等于整数 a，则该表达式就没有必要再重新定义. 这种二义性可以通过说明表达式所在的类集来区别. 如果 x 是"自由的"，则它所在的类集不可能是整数集，但可以是整系数的有理函数集.

一般来讲，用户不希望每一步都说明类集，系统应该尽可能地帮助用户完成这一点. 但类集识别显然不可能完全自动化. 例如：多项式 x^3+x+1 的系数可能属于 Z_2，但该多项式也可能在 $Z_2[x]/(x^4+x^3+1)$ 中. 另一方面，不确切说明对象所属于的类型是十分方便的. 例如：对于一个多项式的系数，我们也许只希望说明它们属于某个交换环. 进一步的信息则可以表示在一个类集与类型的分层结构中 (见 JS).

类集另一个重要方面是储存知识. 我们希望系统知道方程组：

$$\begin{cases} 2x+3y=7 \\ 4x+6y=14 \end{cases}$$

的解空间是一维的，而不仅仅是知道这个方程组，即

方程组：

$2x+3y=7$

$4x+6y=14$

解空间维数 $=1$

[其它性质]

当这附加的信息是通过艰苦的计算得来时，这个特点的意义就更为明显.

AXIOM 系统已经在解释 "整数类集" 和 "最大公因子类集" 方面做出了极有意义的尝试. 例如：我们能写一个一般的求最大公因子的方法. 它对于 α 类的欧基里德环适用 (α 代表计算机代数系统所具有的运算). 应用于 N 时，系统试图把实际的参数看作欧基里德环中的元素 (而不是 Peano 的自然数).

由此引发了对构造性代数分层结构的研究. 它与抽象代数的分层结构不同. 很多研究都集中于这两个分层结构之间是怎样关联的 [DT, DGT].

8.3 重写和自动推导

当我们对三次单位根做运算时， Reduce 中的 Let 规则每次都作 $w^2 \Rightarrow -w-1$ 的化简. 为了防止循环的约化规则，我们通常设定一个全序使得一个项只可能被化为一个比它更低的项. Gröbner 基理论可以看作重写规则的一部分，该理论起源于 Knuth-Bendix[KB]，但 Buchberger 的原始工作比 Knuth-Bendix 的工作更早. 注意，不是总可以得到一组恰当的重写规则. §7.2 的结果说明在多项式情形我们是非常幸运的.

与此相关的领域是半自动推导 (这里 "半" 是因为推导过程基本上是人机对话式的) 和机器验证. 这些领域新的实验性的系统正在不断出现 (例如: 在 Edinburgh 由 R. Pollack 研究的 LEGO, 在巴黎 INRIA 由 T. Coquand 等研制的 COQ). 可以预计会有更多的工具来验证数学教科书中那些可为机器阅读的命题. 这些系统也许可以和计算机代数系统联合应用. 然而, 在计算机代数系统中实现证明验证器的工作至今仍然很少.

8.4 程序块与程序库

几个设计原理, 其中有的来自面向对象的程序, 能帮助研究者将软件块结合起来. 用的方法类似于连接网络中的黑箱子. 其基本方法是通过用户界面将程序模块互相连接, 很象 UNIX 中的 "piping", 并且逐渐演化为用户的显示屏前的图形化的网箱, 其中每一个箱子代表一个具有清晰的输入输出协议的软件包.

正如 §4.3 所述, 这一新技术开发的原因是我们不能期望一个软件包在各方面都是最好的, 另一方面, 为了得到解决手头问题的所有可能的计算工具, 我们仍希望应用最好的软件. 一个通用系统的用户语言所提供的是原始函数 (例如: differentiate, solve 或者 lies in). 一个原始函数也许会利用其它的信息, 从一套程序块中选择出一个最好的. 一个常用的程序块是由 B. McKay 编制的 C 程序 "nauty". 它的用途是决定一个图的所有对称.

运用程序库的另一个原因是提高软件设计者, 而不是用户, 的工作效率. 在计算机上使用一个关于数学的程序库远比为类似的问题写相似的程序方便. Maple 就包含着用 PASCAL 风格写成的数学程序块. 从原则上讲, 这些程序库可以翻译成其它软件包的程序 (例如: Reduce 和 Mathematica). 比利时的 D. Constales 已经开展了程序自动翻译的试验性工作.

8.5 用户界面

最通用的计算机代数系统都有利用 X-Window 的版本, 例如: Reduce 的 MathScribe 和 Maple 的 IRIS.

人们已经设计了工作站上为各种典型的数学活动所服务的用户界面. 例如: CaminoReal, CAS/PI, SUI. 但还没有一个界面可以集中体现新一代用户界面所应该具有的下列功能:

- 阅读百科条目,
- 查询定理, 定义等,
- 阅读定理证明,
- 根据定理进行计算,
- 寻找具有启发性的例子和练习,
- 运用计算机代数系统和其它工具箱,
- 为准备打印数学文章作准备 (例如: TEX码).

- 查询表格和数据库，
- 用某种面向数学的语言写程序，
- 推理和打草稿，
- 观察各种现象的图形，
- 作实验和作图，
- 在不同的阶段运用重写规则，
- 自动检验证明.

回到关于 ab 的例子，在一个超级书写环境下，我们企望用户应该能够按键来发现 a 和 b 的合成的意义，并且在必要时，发现 a 和 b 的类型. 但是，仍需要很大的努力去发现表达和处理数学信息的准确规则.

在 90 年代，为科学和工程计算设计的新的计算机环境已是呼之欲出. 一个比较有效的环境应该是以符号计算的方法为核心，并与绘图工具，数值计算软件库相结合. 计算机代数将在这一环境中唱主角的原因是它更适合于表示和研究数学问题. 这个环境可能是一个工作平台，我们既可以用它做通常的数学工作，也可以为数值计算做准备. 但在这个环境中，我们的工作效率会更高. 我们可以尽可能地利用精确计算，然后通过计算机产生数值计算的 FORTRAN 码，或者直接调用数值计算程序库 (例如 NAG).

9 预测

预测近几年将发生的事情的最好办法是根据正在开展的工作进行推测. 我们将大胆地应用在阿姆斯特丹 CWI 的 Paul Klint 得到的结果. 他宣称需要 15 到 20 年的时间才能完成从概念到应用的过程. 他把这一过程分为三个阶段. 1. 引入想法，2. 该想法被某个研究团体接受，3. 实现该想法对社会的最终应用. 他的一些例子说明了这一观点：

Idea	Intr.	Acc.	Appl.	Total Time elapsed
Unix	70	75	89	19
Smalltalk	72	80	90	18
Ethernet	74	80	89	15
Windows & mouse	70	80	90	20
Prolog	72	78	85	13
Parser generation	70	80	91	21
If-then-else (Fortran)	60	65	77	17
Term rewriting	40–70	80	91	21

表中： Idea, 想法； Intr., 想法的引入； Acc., 想法的接受； Appl., 想法被应用； Total Time elapsed, 所用的时间.

让我们应用这一观点对计算机代数作预测. 根据现在正在进行的工作，我们也许能推断在未来几年中什么样的产品将问世.

Idea	Intr.	Acc.	Appl.	Total Time elapsed
General computer algebra	65	83	88	23
Theorem proving	68	85	95	27
Math. term rewriting	70	85	95	25
Reals as black boxes	75	85	95	20
Object oriented progr.	72	86	95	23
Expert systems	80	90	97	17

表中的 "一般的" 计算机代数 ("general" computer algebra) 指我们上面提到的内容，区别于专门的应用 (如在广义相对论或计算群论中的应用)；在应用领域的计算机代数遵循不同的 (更快的) 演化过程.

表中的前三项说明需要更长的时间使一个数学想法被完全接受. 这也许是因为原始的数学思想与实际应用的距离更远. 有理由相信，计算机代数将缩短这个距离.

与一般的领域不同，我们在表中插入了一个非常具体的条目. 即把实数表示为黑箱子. 在此，我们想到的是用柯西序列逼近实数 α 的算法. 表示 α 的黑箱子将对每一个给定的精度，给出一个在误差范围内的有理数，它可以通过计算机表示.

所谓专家系统，我们是指 §8.5 中提到的用户接口，虽然这不是一个数学对象，但面向对象的程序语言很可能对计算机代数有较大冲击.

假如我们能从这些推测中得出什么结论的话，这个结论就是我们还没有看到计算机代数发展的终点.

参考文献： (略)

(李子明 译　戴宗铎 校)