计算机代数系统
数学原理

李超 阮威 张龙 张翔

maTHμ Project Group

www.mathmu.cn

清华大学

2009年8月17日

摘要

本文主要讨论计算机代数系统的数学原理, 由十六个章节组成. 内容包含高精度运算, 数论, 数学常数, 精确线性代数, 多项式, 方程求解, 符号求和, 符号积分, 微分方程符号解等九大部分, 涵盖了构建计算机代数系统的最基础也是最重要的内容. 许多内容是第一次被系统地整理出现在中文文献中, 一些领域也追踪到了最新进展.

绪言

计算机代数(Computer Algebra)在很多时候又被广义地理解为“符号计算”(Symbolic Computation), 成为与所谓“数值计算”(Numerical Computation)相对的概念. “符号”的运算在这里代替了“数”的运算. 这是一种智能化的计算. 符号可以代表整数, 有理数, 实数和复数, 也可以代表多项式, 函数, 还可以代表数学结构如集合, 群, 环, 代数等等. 我们在学习和研究中用笔和纸进行的数学运算多为符号运算.

利用计算机代数, 我们可以完成许多不可思议的事情, 例如可以对代数方程组进行精确的求解, 对多项式进行因子分解, 对复杂代数表达式进行化简归约, 对函数进行符号积分(求出原函数), 对微分方程求出精确解等等.

传统的代数计算冗长繁杂, 而现代的计算机技术为大型的符号计算提供了可能性. 关键的问题就在于如何把抽象的代数理论算法化, 使其高效地处理形形色色的代数问题. 强大的计算机代数系统不仅是各类工程技术的助手, 对纯粹科学研究也起着不可忽略的推动作用.

经过数十年的发展, 在国外已经形成了诸如 Wolfram Research[1], Maplesoft[2] 等巨型的商业软件公司, 其产品具有可观的经济效益; 其他一些研究者的专用系统开发也具有了相当的规模. 然而, 在我们国内, 科学软件领域则远远落后, 能够与国外产品相抗衡的通用计算机代数系统还暂时没有出现. 而另一方面, 国内对科学软件的需求量却是巨大的, 昂贵的进口产品意味着大量的科研, 工程经费的无奈外流. 从某种意义上来说, 对国外系统的依赖对国家信息安全也有着潜在的威胁.

在我们看来, 造成这种状况的原因一方面是由于科学软件的复杂性, 另一方面则反映了国内创新能力的缺乏. 就这一点来说, 国内的大环境是一个很重要的因素:

- 在国内盗版软件获取容易, 而大型的科学软件开发却需要长期, 大量的投入, 其复杂程度及难以预料的经济前景使得企业少有问津;

[1] http://www.wolfram.com
[2] http://www.maplesoft.com

- 一个更重要的原因是, 在中国往往出现这种情况: 软件写得顺手的人, 未必有很强的科学计算背景, 而科学背景强的人又没有较高的计算机和软件水平, 使得科学软件的开发难以进行;

- 少有"不切实际的幻想". 由于科学软件的复杂程度, 企业少有问津, 单独的科研人员也不得不认为完成这样一件工作是"不切实际的幻想". 然而我们回顾 Wolfram Research, Maplesoft, Mathworks 等大公司的发展道路, 无不是从一个人或几个人的的微薄之力逐渐发展到现在令人赞叹的规模. 或许"不切实际的幻想"正是创新氛围的一个绝佳体现吧!

尽管我们力量渺小, 但对这种状况也不愿意置若罔闻. 清华大学作为一所综合大学, 众多学子具有综合性的能力, 也理应在此领域有所作为. 我们希望利用自身的数学基础与应用背景, 整理出一份较为完整的计算机代数理论文档, 进行较为完整系统设计, 并实现一个计算机代数系统核心 maTHμ[3]. 希望在此过程中也能广阔我们的眼界, 提高我们自身的理论和技术水平, 为更进一步的工作打下良好基础.

这份计算机代数系统数学原理, 作为 maTHμ 系统的理论文档, 便是 maTHμ 项目组成员通过近一年半时间集体整理撰写的成果. 计算机代数理论方面的中文文献稀缺. 即使在英文文献中, 能够足以支撑一个通用计算机代数系统的系统论述也少见, 更多的内容散见在专著, 博士论文及专业期刊中. 花大力气整理文档的初衷十分简单: 理论不清, 则后续的设计与开发阶段根本无法进行, 更何况计算机代数理论本身复杂而相互交织. 举例来说, 求微分方程的符号解需要符号积分的支持, 而有理函数符号积分需要借助精确线性代数, 多项式因子分解, 代数方程组求解等算法, 其中 $\mathbb{Z}[x]$ 上的多项式因子分解依赖有限域上的因子分解, 从而需要数论中模算法的支持, 而数论算法又建立在任意精度的快速运算上. 整个文稿的内容组织也大体符合构建计算机代数系统的逻辑顺序.

本文中大部分的算法都有理论的推导, 努力做到自成体系并阐明各种方法背后的想法. 对于若干深刻的结果(如 Hilbert 零点定理, Liouville 定理, Lie-Kolchin 定理等等), 鉴于篇幅和目的, 我们选择只给出相关参考文献而略去了严格证明.

本文中的部分算法已经在 maTHμ 1.0 版的内核中实现. 计算机代数系统 maTHμ 项目在 2009 年清华大学第二十七届"挑战杯"课外学术科技竞赛中, 经过 4 轮评审, 最终在 344 件作品中脱颖而出, 获得了特等奖的第一名, 并代表清华大学参加两年一度的全国"挑战杯"竞赛. 项目未来的长期发展规划也已经得到了学校的大力支持. 项目团队感受到了来自各方的鼓励与期望, 希望能踏实地继续努力进行我们的工作. 在理论文档部分, 除了继续丰富与完善呈现在这里的内容, 其他

[3] http://www.mathmu.cn

一些我们认为同样重要的主题也正在整理中, 包括初等与特殊函数的任意精度计算, 组合函数, 代数函数积分, 更一般的符号求和理论, 表达式化简等等.

作者 于清华园
maTHμ Project Group
maTHmU@gmail.com
2009年8月17日

目录

1 高精度运算

我们所熟知的科学计算一般就是指数值计算. 数值计算是计算数学的一个主要部分, 它研究用计算机求解各种数学问题的数值计算方法及其理论与软件实现. 关于数值计算的研究在计算机被发明之前就已经有了相当的基础, 它涉及到的内容包括函数的数值逼近, 数值微分与数值积分, 非线性方程数值解, 数值线性代数, 常微分方程与偏微分方程数值解等(参见 [7]). 数值计算中处理的对象并不仅仅是数值, 还包括由数值构成的简单数据结构, 例如一般的多项式, 无穷级数, 矩阵等, 数值计算处理问题的一般方法是通过数学推导将问题化归到这些数学对象的运算上.

作为应用数学, 数值计算的主要目标是解决来自于生产实践的工程学问题. 与此同时, 数学工作者做数学研究本身也是一种生产实践, 数学研究过程中同样会产生许多问题, 与工程学问题不同, 这些问题往往只能用抽象的符号来表达, 仅用数值计算的方法是不易解决的, 对于这类问题解决方案的研究逐渐形成了应用数学的一个新的分支, 为了与数值计算相区别, 常常称之为符号计算. 类似地, 我们可以给符号计算下一个简单的定义:符号计算是一门研究用计算机求解各种数学问题的符号计算方法及其理论与软件实现的科学, 它是数学(家)的计算数学. 符号计算中处理的数据和结果都是符号, 这种符号可以是字母, 公式, 也可以是数. 与数值计算不同的是, 数是作为一种符号出现在符号计算中的, 这就要求关于数的运算应该是绝对精确的, 我们接下来就要讨论数的高精度运算.

1.1 整数

在基于硬件的整数指令中, 计算机能够处理的整数是有界的, 在目前典型的计算机中整数的溢出界都不超过 2^{64}, 而符号计算中常常需要处理更大的整数, 例如阶乘, Fibonacci 数列这样简单的数论函数计算. 另一个不平凡的例子是所谓的中间表示膨胀(intermediate expression swell)(参见 [14] 第 2 章), 例如采用 Euclid 算法计算两个整系数多项式的最大公因子时, 即使输入的两个多项式和输出的最大公因子都具有绝对值较小的系数, 计算过程中的中间结果仍然很可能出现绝对值非常大的系数. 设

$$F = 7x^7 + 2x^6 - 3x^5 - 3x^3 + x + 5,$$
$$G = 9x^5 - 3x^4 - 4x^2 + 7x + 7,$$

在计算过程中将有理数化为整数, 我们将得到如下的多项式序列

$$1890x^4 - 4572x^3 - 6930x^2 - 846x + 4527,$$
$$294168996x^3 + 257191200x^2 - 20614662x - 142937946,$$
$$- 1036852783698413052 00x^2 - 32576054233115610000x$$
$$+ 122463167842311670000,$$
$$29567908335038495467893420575652070982917635200 00x$$
$$+ 55532526180624799696603478407402529168762016 0000,$$
$$10920746857330312192010416027912598626591699661845938035 18$$
$$60241877714068288433476964706354360773769842688000000000 0,$$

最后的那个整数达到了 118 位. 除此之外, 高精度浮点数的表示和运算也是直接依赖于高精度整数的.

1.1.1 进制转换

为了提高运算效率, 高精度整数的内部表示常常采用 2 的正整数次幂进制, 如 2^{32} 进制或 2^{64} 进制, 而人们书写或阅读时更习惯于采用十进制, 因此高精度整数输入输出时常常需要做进制转换. 即给定正整数 n 的 B 进制表示

$$n = (a_s \dots a_1 a_0)_B,$$

需要得到 n 在 B' 进制下的表示.

进制转换的计算方法一般有如下两种.

算法1.1 (在 B 进制下除以 B').

输入:正整数 n 的 B 进制表示 $n=(a_s\dots a_1a_0)_B$.

输出:n 的 B' 进制表示.

1. 设 n 的 B' 进制表示为 $(b_t\dots b_1b_0)_{B'}$.
2. 在 B 进制下依次计算出
$$b_0=n \bmod B', b_1=\left[\frac{n}{B'}\right] \mod B', \dots.$$
3. 返回 $(b_t\dots b_1b_0)_{B'}$.

算法1.2 (在 B' 进制下乘以 B).

输入:正整数 n 的 B 进制表示 $n=(a_s\dots a_1a_0)_B$.

输出:n 的 B' 进制表示.

1. 设 B' 进制整数 $x=0$.
2. 利用 Horner 法则(参见 [181]), 在 B' 进制下依次计算 $x=Bx+a_k$, k 从 s 到 0.
3. 返回 x.

注1. 当待转换的数 n 较长时, 进制转换应该分段进行. 以"在 B 进制下除以 B'"的转换算法(算法 1.1)为例, 设待转换的数为 n, 先反复地用 B'^m 除 n 从而得到 n 的 B'^m 进制表示, 然后将 n 的 B'^m 进制表示的每一位再反复地用 B' 除从而转换成 m 位 B' 进制数字. 这样做的好处是第二步除以 B' 的除法一般只是单精度运算, 因此大大节省了运算时间, 关于这一点更详细的介绍请参阅 [104].

1.1.2 四则运算

本章的开头曾经强调过, 数是作为一种符号出现在符号计算中的, 这就要求关于数的运算应该是绝对精确的. 在机器精度的范围内, 现有的计算机可以轻松地完成整数的四则运算, 这主要得益于计算机设计师们的工作. 而接下来要讨论的高精度整数四则运算, 其基本原理其实和基于硬件的整数指令是一致的, 事实上, 这就是阿拉伯计数法的发明者们很早就发展出的一整套借助纸笔进行的四则运算理论,

我们从小就开始学习熟练地利用它们来操作整数. 加法, 减法和普通乘法是平凡的, 这里不再赘述, 而重点讨论一下除法.

商为一位数的除法

在四则运算的笔算方法中, 除法可能是最复杂的. 因为除法需要试商, 试商包含着猜测的成分, 于是不容易形成有效的算法. 机器精度整数除法也要试商, 但由于计算机内部采用了二进制表示, 实际上每一次试商的结果都只有 0, 1 两种可能, 因此只需要比较相除的两个数的大小就可以确定. 为了得到高精度整数除法的有效算法, 我们必须要将一般的试商过程算法化.

首先来看商为一位数的除法, 即假定商 $\left[\frac{a}{b}\right]$ 满足

$$0 \leq \left[\frac{a}{b}\right] \leq B-1.$$

笔算除法的经验告诉我们, 被除数和除数的最高几位数对试商是很重要的, 我们常常凭借目测(口算)最高几位数就能基本上把商给确定下来.

定理1.1. 设整数 $a=(a_{s+1}a_s\dots a_1a_0)_B, b=(b_s\dots b_1b_0)_B$, 则商 $\left[\frac{a}{b}\right]$ 满足不等式

$$\left[\frac{a_{s+1}B+a_s}{b_s+1}\right] \leq \left[\frac{a}{b}\right] \leq \min\left\{\left[\frac{a_{s+1}B+a_s}{b_s}\right], B-1\right\}.$$

证明. 首先

$$\frac{a}{b} > \frac{a_{s+1}B^{s+1}+a_sB^s}{(b_s+1)B^s} = \frac{a_{s+1}B+a_s}{b_s+1},$$

故 $\left[\frac{a_{s+1}B+a_s}{b_s+1}\right] \leq \left[\frac{a}{b}\right]$, 不等式的前半部分得证.

而由

$$\left[\frac{a_{s+1}B+a_s}{b_s}\right] > \frac{a_{s+1}B+a_s}{b_s}-1$$

可以得到

$$\begin{aligned}\left[\frac{a_{s+1}B+a_s}{b_s}\right]+1 &\geq \frac{a_{s+1}B+a_s+1}{b_s}\\ &= \frac{a_{s+1}B^{s+1}+(a_s+1)B^s}{b_sB^s}\\ &> \frac{a}{b},\end{aligned}$$

故 $\left[\frac{a}{b}\right] \leq \left[\frac{a_{s+1}B+a_s}{b_s}\right]$, 证毕. □

注2. 记

$$q = \min\left\{\left[\frac{a_{s+1}B+a_s}{b_s}\right], B-1\right\},$$

则 q 是商 $\left[\frac{a}{b}\right]$ 一个很好的上界, 事实上下面的定理将告诉我们 q 比商的真值至多大 2.

定理1.2. *若 b 的最高位不小于 $\left[\frac{B}{2}\right]$, 即 $b_s \geq \left[\frac{B}{2}\right]$, 则*

$$q-2 \leq \left[\frac{a}{b}\right] \leq q.$$

证明. 用反证法. 假设 $\left[\frac{a}{b}\right] < q-2$, 则

$$\left[\frac{a}{b}\right] \leq \left[\frac{a_{s+1}B+a_s}{b_s}\right] - 3, \quad \left[\frac{a}{b}\right] \leq B-4.$$

因为

$$\left[\frac{a}{b}\right] > \frac{a}{b} - 1, \quad \frac{a_{s+1}B+a_s}{b_s} < \frac{a}{b-B^s},$$

所以 $\frac{a}{b}+2 < \frac{a}{b-B^s}$, 推出

$$\frac{a}{b} > 2(\frac{b}{B^s}-1) \geq 2(b_s-1).$$

因此 $b_s \leq \frac{B}{2}-1 < \left[\frac{B}{2}\right]$, 与题设条件矛盾, 证毕. □

上面只考虑了被除数的头两位与除数的首位, 如果允许做更精细的考察, 譬如说考虑被除数的头三位与除数的头两位, 我们还可以得到对商更好的估计.

定理1.3. *设 $t = (a_{s+1}B+a_s) - qb_s$, 若 $B \cdot t \geq qb_{s-1} - a_{s-1}$, 则*

$$q-1 \leq \left[\frac{a}{b}\right] \leq q.$$

证明. 将 t 代入不等式中得

$$B \cdot ((a_{s+1}B+a_s) - qb_s) \geq qb_{s-1} - a_{s-1},$$

化简得到

$$\begin{aligned} a &\geq a_{s+1}B^2 + a_sB + a_{s-1} \\ &\geq q(b_sB + b_{s-1}) \\ &> q(b - B^{s-1}). \end{aligned}$$

因为 $q(b-B^{s-1}) > qb - B^s \geq (q-1)b$, 所以 $\frac{a}{b} > q-1$, 证毕. □

注3. 如果定理中的条件不满足, 将 q 减 1, 这时我们总可以说 q 比商的真值至多大 1.

综合以上这些想法, 现在可以写出商为一位数的除法的详细过程了.

算法1.3 (商为一位数的除法)**.**

输入:整数 $a=(a_{s+1}\dots a_1a_0)_B$, $b=(b_s\dots b_1b_0)_B$.

输出:a,b 的商 $[\frac{a}{b}]$.

1. 若 $b_s<\left[\frac{B}{2}\right]$, a,b 同乘以 $\left[\frac{B}{b_s+1}\right]$.
2. 计算
$$q=\min\left\{\left[\frac{a_{s+1}B+a_s}{b_s}\right],B-1\right\}.$$
3. 计算 $t=(a_{s+1}B+a_s)-qb_s$, 如果 $B\cdot t<qb_{s-1}-a_{s-1}$, q 减一.
4. 计算余数 $r=a-qb$, 如果 $r\geq 0$, 返回 q, 否则返回 $q-1$.

整数除法

注意到

$$r=a-qb=a \bmod b$$

在"商为一位数的除法"算法(算法 1.3)的最后一步也同时被算出, 以此为基础可以直接写出一般的商为多位数的除法算法.

算法1.4 (整数除法)**.**

输入:整数 $a=(a_m\dots a_1a_0)_B$, $b=(b_n\dots b_1b_0)_B$.

输出:a,b 的商 $\left[\frac{a}{b}\right]=(q_{m-n}\dots q_1q_0)_B$.

定义辅助序列

$$r_k=\left[\frac{a}{B^k}\right] \bmod b,\quad m-n+1\geq k\geq 0.$$

按递推公式依次求出 $q_k,r_k(m-n+1\geq k\geq 0)$:

1. $q_k=\left[\frac{B\cdot r_{k+1}+a_k}{b}\right]$,
2. $r_k=(B\cdot r_{k+1}+a_k) \bmod b$.

注4. 为了利用快速乘法, 还可以利用 Picarte 迭代来计算整数除法, 具体来说就是先利用数值计算中浮点数的 Picarte 迭代求出除数倒数具有一定精度的近似值, 然

后利用整数乘法将被除数乘上去, 当除数的位数较多时, 这种方法是很有效的, 关于这一点更详细的介绍请参阅 [81].

1.2 快速乘法

高精度整数是计算机代数系统的内置基本类型, 其四则运算的快慢对系统的性能好坏有着决定性的影响. 目前最著名的高精度整数运算库是 GNU 的 GMP[167], 许多著名的计算机代数系统如 Axiom, Maple, Mathematica, Maxima 等的底层高精度整数运算都是基于 GMP 实现的(参见 [168]). 加法和减法的复杂度关于整数位数是线性的, 考虑到输入输出的复杂度关于整数位数也是线性的, 因此从算法上来看加减法已经达到了复杂度的下界. 而高精度除法总可以通过 Picarte 迭代归结为高精度乘法(注 4), 所以高精度四则运算的主要问题集中到了乘法上.

设 n 为乘数的位数, 就目前已知的情况而言, 不同乘法算法的时间复杂度可以从平凡的 $O(n^2)$(普通乘法), $O(n^{\log_2 3})$(Karatsuba 乘法), $O(n^{\log_3 5})$(Toom-3 乘法), $O(n\log^* n)$(复数域上的 FFT), 其中

$$\log^* n = \log n(\log\log n)(\log\log\log n)\cdots,$$

和 $O(n(\log n)(\log\log n))$(有限域上的 FFT), 其中“有限域上的 FFT”的时间复杂度已经相当接近线性了, [104] 和 [174] 中给出了具体的复杂度分析与证明. 但是这些乘法算法中复杂度较低的算法往往有较大的常数因子, 因此如果乘数的位数较少, 普通乘法反而是最快的, 所以实用中常常将这些不同的乘法算法结合起来使用, 每次做乘法时都根据相乘两数的大小动态地选择具体采用哪一种算法, 而每种算法的最佳适用范围往往依赖于具体实现和硬件环境, 因此一般直接通过实验来确定.

1.2.1 一元多项式乘法

单从整数乘法的角度来看, 笔算乘法的算法是如此直截了当, 以至于很难想象出更好的算法. 因此我们换一个角度, 先来考虑与整数乘法具有很大相似性的一元多项式乘法, 这里需要用到一元多项式的系数表示和点值表示的概念.

定义1.1 (一元多项式系数表示)**.** 一元多项式

$$A(x) = \sum_{k=0}^{n-1} a_k x^k,$$

的系数表示就是一个由系数组成的向量 $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})$.

注5. 我们习惯的表示多项式的方式其实就是系数表示, 每个多项式的系数表示向量都是唯一确定的.

定义1.2 (一元多项式点值表示)**.** 一元多项式

$$A(x)=\sum_{k=0}^{n-1}a_kx^k,$$

的点值表示是 $A(x)$ 在 n 个不同点处的点值对构成的集合

$$\{(x_0,y_0),(x_1,y_1),\ldots,(x_{n-1},y_{n-1})\},$$

其中 $y_k=A(x_k),k=0,1,\ldots,n-1$.

注6. 一个多项式可以有很多种不同的点值表示, 这是因为每一组 x_k 都决定着一种点值表示.

一元多项式的系数表示与点值表示之间是可以相互转换的, 系数表示到点值表示的转换就是一元多项式多点求值, 而点值表示到系数表示的转换就是一元多项式插值. 关于这两种转换的高级讨论, 如一元多项式快速多点求值, 一元多项式快速插值等, 请参阅“多项式快速求值与插值”一章.

两个一元多项式的乘积有如下的系数表示, 这可以从一元多项式乘积的定义直接得到.

定理1.4. 已知两个次数小于 n 的多项式的系数表示分别为

$$A(x)=\sum_{k=0}^{n-1}a_kx^k,\quad B(x)=\sum_{k=0}^{n-1}b_kx^k,$$

设乘积的系数表示为 $C(x)=\sum\limits_{k=0}^{2n-2}c_kx^k$, 那么

$$c_k=\sum_{i+j=k}a_ib_j.$$

注7. 根据定理 1.4 来直接计算一元多项式的乘积时总共需要 n^2 次系数乘法. 现在我们首先取定 $2n-1$ 个不同点 x_k, 然后利用多项式多点求值计算出 $A(x)$ 与 $B(x)$ 在这组点上的点值表示, 如果 $C(x)=A(x)\cdot B(x)$, 那么显然有 $C(x_k)=A(x_k)\cdot B(x_k)$, 只需要 $2n-1$ 次系数乘法就可以计算出乘积 $C(x)$ 的点值表示, 再通过多项式插值就可以得到 $C(x)$ 的系数表示.

注 7 表明利用多项式的点值表示也可以做多项式乘法, 而且过程显得更简洁. 如果多项式的默认表示形式就是点值表示, 那么连多项式多点求值和多项式插值这两个转换步骤也可以省去了, 多项式乘法就和向量逐点相乘一样简单.

整数与多项式之间的相似性是由整数的进制表示

$$(a_n \dots a_1 a_0)_B = \sum_{k=0}^{n} a_k \cdot B^k$$

产生的, 与整数的进制表示相对应的是多项式的系数表示. 我们要从注 7 中的多项式乘法算法中产生整数快速乘法算法, 就必须解决多项式系数表示与点值表示之间的转换问题, 因此接下来的要介绍整数快速乘法算法也就自然包含多项式多点求值, 向量逐点相乘, 多项式插值这三个步骤.

1.2.2 Karatsuba 乘法

第一个不平凡的快速乘法算法是由俄罗斯数学家 A. Karatsuba 于 1962 年发现的(参见 [99]), 它采用了一个简单但十分有效的分治策略来使乘法加速. Karatsuba 乘法 实现起来并不困难, 因此在算法理论中也常常被拿来当作分治算法(参见 [60])的很好例子.

一次多项式的乘法

为了更多地了解隐藏在算法背后的思想, 在介绍 Karatsuba 算法之前, 我们先看一看如何利用多项式的点值表示来计算一次多项式的乘法, 这是一个很具有启发性意义的例子.

定义1.3. 多项式 $f(x)$ 在 ∞ 处的值定义为

$$f(\infty) = \lim_{x\to\infty} f(x)/x^{\deg f}.$$

注8. $f(\infty)$ 其实就等于 $f(x)$ 的首项系数, 可以证明这样的定义与多项式插值是相容的.

例1.1. 设

$$A(x) = a_1 x + a_0, \quad B(x) = b_1 x + b_0,$$

选取插值点组为 $x_0 = -1, x_1 = 0, x_2 = \infty$, 则 $A(x), B(x)$ 的点值表示分别为 $\{(-1, a_0 - a_1), (0, a_0), (\infty, a_1)\}$, $\{(-1, b_0 - b_1), (0, b_0), (\infty, b_1)\}$, 如果 $C(x) = A(x) \cdot B(x)$, 那么 $C(-1) = (a_0 - a_1) \cdot (b_0 - b_1)$, $C(0) = a_0 \cdot b_0$, $C(\infty) = a_1 \cdot b_1$, 利用简单的多项式插值算法, 可以计算出 $C(x)$ 的系数表示

$$(c_0, c_1, c_2) = (C(0), C(0) + C(\infty) - C(-1), C(\infty)),$$

即

$$C(x) = a_1b_1x^2 + (a_0b_0 + a_1b_1 - (a_0 - a_1)(b_0 - b_1))x + a_0b_0.$$

注9. 在这个例子中, 多项式多点求值与多项式插值都只用到系数加减法, 这表明可以只用三次系数乘法完成一次多项式的乘法.

注10. 如果 a, b 都是 $2n$ 位 B 进制整数, 那么在 B^n 进制下 a, b 都是两位数

$$a = a_1B^n + a_0, \quad b = b_1B^n + b_0,$$

其中 $a_i, b_i (i = 0, 1)$ 都是 B^n 进制下的个位数, 也就是 B 进制下不超过 n 位的数. 那么根据例 1.1 我们有

$$a \cdot b = a_1b_1B^{2n} + (a_0b_0 + a_1b_1 - (a_0 - a_1)(b_0 - b_1))B^n + a_0b_0.$$

Karatsuba 乘法

根据注 10 , 我们现在写出 Karatsuba 乘法算法的详细过程.

算法1.5 (Karatsuba 乘法)**.**

输入:n_a 位整数 a 和 n_b 位整数 b.

输出:a, b 的积 $c = a \cdot b$.

1. 令 $n = \max\{n_a, n_b\}$, 若 $n = 1$, 利用普通乘法计算并返回 $a \cdot b$;若 n 为奇数, 令 $n = \frac{n+1}{2}$, 否则令 $n = \frac{n}{2}$.
2. 若 $n_a \le n$, 令 $a_1 = 0, a_0 = a$;否则令 $a_1 = a$ 的高 $n_a - n$ 位, $a_0 = a$ 的低 n 位.
3. 若 $n_b \le n$, 令 $b_1 = 0, b_0 = b$;否则令 $b_1 = b$ 的高 $n_b - n$ 位, $b_0 = b$ 的低 n 位.
4. 使用 Karatsuba 乘法计算 $c_0 = a_0b_0, c_1 = a_1b_1, c_2 = (a_0 - a_1)(b_0 - b_1)$, 返回

$$(c_1 \cdot B^{\frac{n}{2}} + c_0 + c_1 - c_2) \cdot B^{\frac{n}{2}} + c_0.$$

注11. 与普通乘法类似, 在第 4 步中乘上 B^k 时只需要在右边添上 k 个 0, 或者说左移 k 位.

记 $T(n)$ 是递归地利用 Karatsuba 乘法将两个 n 位 B 进制数相乘所需要的 B 进制个位数乘法次数, 那么有 $T(2n) \le 3T(n)$, 由此推出 $T(n) = O(n^{\log_2 3}) \approx O(n^{1.59})$.

关于 Karatsuba 算法更详细的介绍请参阅 [14] 第 2 章, [174], [104] 和 A. Karatsuba 的原始论文 [99].

值得一提的是, 采用和 Karatsuba 乘法完全类似的想法, 还可以得到一种整数除法的快速算法(参见 [49]).

1.2.3 Toom-Cook 乘法

Toom-3 乘法

沿着 Karatsuba 的思路, 考虑二次多项式的乘法.

例1.2. 设

$$A(x) = a_2x^2 + a_1x + a_0, \quad B(x) = b_2x^2 + b_1x + b_0,$$

选取插值点组为

$$x_0 = -1, x_1 = 0, x_2 = 1, x_3 = 2, x_4 = \infty,$$

如果 $C(x) = A(x) \cdot B(x)$, 类似地, 利用简单的多项式插值算法, 可以计算出 $C(x)$ 的系数表示 $(c_0, c_1, c_2, c_3, c_4)$, 其中

$$\begin{aligned}
c_0 &= C(0),\\
c_1 &= C(-1) + C(0) + C(1) + C(2) + C(\infty),\\
c_2 &= C(-1) + C(0) - C(1) - C(2) + C(\infty),\\
c_3 &= 4C(-1) + C(0) + 2C(1) + 8C(2) + 16C(\infty),\\
c_4 &= C(\infty).
\end{aligned}$$

注12. 如果不计乘以较小常数的乘法和加减法, 我们可以只用 5 次系数乘法完成二次多项式的乘法.

根据例 1.2 并仿照 Karatsuba 乘法而得到的算法称为 Toom-3 乘法. 记 $T(n)$ 是递归地利用 Toom-3 乘法将两个 n 位 B 进制数相乘所需要的 B 进制个位数乘法次数, 那么有 $T(3n) \le 5T(n)$, 由此推出 $T(n) = O(n^{\log_3 5}) \approx O(n^{1.46})$.

Toom-r 乘法

更一般地, 考虑 r 次多项式的乘法.

例1.3. 设

$$A(x) = a_rx^r + \cdots + a_0, \quad B(x) = b_rx^r + \cdots + b_0,$$

选取插值点组为

$$x_0 = -r, x_1 = -(r-1), \ldots, x_r = 0, x_{r+1} = 1, \ldots, x_{2r} = r,$$

设

$$C(x) = A(x) \cdot B(x) = \sum_{k=0}^{2r} c_k x^k,$$

则

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{2r} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & \cdots & x_0^{n-1} \\ 1 & x_1 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \ldots & \vdots \\ 1 & x_{2r} & \cdots & x_{2r}^{2r} \end{bmatrix}^{-1} \begin{bmatrix} A(x_0) \cdot B(x_0) \\ A(x_1) \cdot B(x_1) \\ \vdots \\ A(x_{2r}) \cdot B(x_{2r}) \end{bmatrix}.$$

与 Toom-3 乘法的情形类似, 根据例 1.3 对 Karatsuba 乘法做推广就可以得到 Toom-r 乘法. 记 $T(n)$ 是递归地利用 Toom-r 乘法将两个 n 位 B 进制数相乘所需要的 B 进制个位数乘法次数, 那么有 $T(rn) \leq (2r+1)T(n)$, 由此推出 $T(n) = O(n^{\log_{r+1} 2r+1})$.

这样得到的一系列快速乘法算法统称为 Toom-Cook 乘法, 关于 Toom-Cook 乘法更详细的介绍请参阅 [104]. Toom-Cook 乘法中的 Toom-2 乘法与 Karatsuba 乘法基本相同, 只是选取的 3 个插值点不一样.

由于

$$\lim_{r \to \infty} \log_{r+1} 2r + 1 = 1,$$

所以使用 Toom-Cook 乘法进行 n 位整数乘法所需要的 B 进制个位数乘法次数的下界趋近于 $O(n)$, 这是一个很好的结果. 但是由于多项式多点求值和多项式插值所需要的线性级操作会随着 r 的增加而迅速增加, 以至于一般说来仅仅当 $r = 2, 3$ 时, Toom-Cook 乘法才是实用的.

1.2.4 FFT 乘法

FFT(Fast Fourier Transform)常常被认为是 20 世纪数值计算和算法领域最重要的成果之一, 它可以快速计算向量卷积, 这一点对于信号处理等工程领域尤其重要. 对于符号计算来讲, FFT 实际上可以看成是多项式快速多点求值和快速插值算法一个高度优化的特例. 在这里我们只需要利用多项式乘法和向量卷积之间的相似性, 就可以很好地利用 FFT 来加速多项式乘法和整数乘法.

在上一节中, Toom-Cook 乘法通过充分地利用分治与递归有效地减少了多项式多点求值和多项式插值操作的消耗;下面将要讨论的 FFT 乘法则是通过精心地挑选求值点, 把系数表示与点值表示之间的转换所需的操作压缩为次线性级别, 即 $O(n \log n)$, 其中的 n 是乘数的位数.

DFT

顾名思义, FFT 是快速计算 DFT(Discrete Fourer Transform)的算法, 我们可以将 DFT 简单的理解成以单位复根作为求值点时多项式系数表示到点值表示之间的转换. 在正式给出 DFT 的定义之前, 需要先引入原根和循环卷积的概念.

定义1.4 (原根)**.** 设 $n \in \mathbb{N}_+$, R 是一个环, $\omega \in R$.

1. 若 $\omega^n = 1$, 则称 ω 为 R 的 n 次单位根.
2. 若 ω 是 n 次单位根, 并且对于 n 的每个素因子 p, $\omega^{n/p} - 1$ 不是 R 的零因子, 则称 ω 为 R 的 n 次单位原根.

注13. 按照这个定义, 任意素数的整数次幂单位复根都是复数域 $\mathbb{C}$ 的单位原根, 例如 $\omega = e^{\frac{2\pi i}{8}} \in R = \mathbb{C}$ 就是一个 8 次单位原根. 除此之外, 对于模整数环 $\mathbb{Z}_{17}$, $\overline{3} \in R = \mathbb{Z}_{17}$ 是 16 次单位原根, 与此同时, 虽然 $\overline{2}^{16} = \overline{1} \in R = \mathbb{Z}_{17}$, 但 $\overline{2}$ 并不是 $\mathbb{Z}_{17}$ 的 16 次单位原根, 因为 $2^8 - 1 \equiv 0 \mod 17$. 如果 q 是素数的整数次幂, 那么当且仅当 $n|q-1$ 时有限域 $\mathbb{F}_q$ 存在 n 次单位原根.

关于原根有如下定理.

定理1.5. *如果 ω 是环 R 的 n 次单位原根, 那么 $\forall 1 < k < n$, $\omega^k - 1$ 不是 R 的零因子, 并且*

$$\sum_{j=0}^{n-1} \omega^{jk} = 0.$$

证明. 设 $d = \gcd(k, n)$ 且 $u \cdot k + v \cdot n = d$, 由 $d < n$ 知存在 n 的一个素因子 p 使得 $d|n/p$. 从而 $\omega^d - 1|\omega^{n/p} - 1$, 但 $\omega^{n/p} - 1$ 不是 R 的零因子, 因此 $\omega^d - 1$ 也不是 R 的零因子.

与此同时,

$$\omega^k - 1|\omega^{u\cdot k} - 1 = \omega^{u\cdot k + v\cdot n} - 1 = \omega^d - 1,$$

这就证明了 $\omega^k - 1$ 不是 R 的零因子.

由 $\omega^n = 1$ 知

$$0 = \omega^{kn} - 1 = (\omega^k - 1) \cdot (\sum_{j=0}^{n-1} \omega^{jk}),$$

而 $\omega^k - 1$ 不是 R 的零因子, 因此

$$\sum_{j=0}^{n-1} \omega^{jk} = 0.$$

□

定义1.5 (循环卷积)**.** 设 $f(x)=\sum\limits_{k=0}^{n-1}f_kx^k, g(x)=\sum\limits_{k=0}^{n-1}g_kx^k$, 则称 $f(x)*g(x)=\sum\limits_{k=0}^{n-1}h_kx^k$ 为 $f(x),g(x)$ 的循环卷积, 其中 $h_k=\sum\limits_{i+j\equiv k \bmod n}f_ig_j$.

注14. 不难发现, 循环卷积的定义跟多项式乘积的定义很相似, 我们可以简单地把它理解成先扩充多项式次数再做多项式乘法的过程. 设 f,g 的多项式乘积 fg 是 $n-1$ 次多项式, 则 f 和 g 的次数可能小于 $n-1$, 如果在它们的最高次项之前添上系数为 0 的高次项并把它们看成 $n-1$ 次多项式, 这时候会发现 f,g 的循环卷积 $f*g$ 恰好等于 fg, 这就是循环卷积和普通多项式乘积之间的关系.

现在可以正式地给出 DFT 的定义了.

定义1.6 (DFT)**.** 设 $f(x)=\sum\limits_{k=0}^{n-1}f_kx^k$, ω 是环 R 的 n 次单位原根, 则称

$$\mathrm{DFT}_\omega(f)=(f(1),f(\omega),\ldots,f(\omega^{n-1}))$$

为 f 的离散傅立叶变换.

DFT 存在逆变换, 并且原根的良好性质(定理 1.5)保证了 DFT 的逆变换和 DFT 具有相同的结构.

定理1.6. *设 ω 是环 R 的 n 次单位原根, 用 $(\mathrm{DFT}_\omega)^{-1}$ 表示 DFT_ω 的逆变换, 那么*

$$(\mathrm{DFT}_\omega)^{-1}=\frac{1}{n}\mathrm{DFT}_{\omega^{-1}}.$$

证明. DFT_ω 是 R^n 上的线性变换, 它在自然基下的矩阵表示为 $\vec{y}=V_n(\omega)\vec{a}$, 即

$$\begin{bmatrix}y_0\\y_1\\y_2\\\vdots\\y_{n-1}\end{bmatrix}=\begin{bmatrix}1&1&\cdots&1\\1&\omega&\cdots&\omega^{n-1}\\1&\omega^2&\cdots&\omega^{2n-2}\\\vdots&\vdots&&\vdots\\1&\omega^{n-1}&\cdots&\omega^{(n-1)^2}\end{bmatrix}\begin{bmatrix}a_0\\a_1\\a_2\\\vdots\\a_{n-1}\end{bmatrix},$$

其中 $V_n(\omega)$ 为 n 阶 Vandermonde 方阵. $V_n(\omega)$ 的 (i,j) 位元素是 ω^{ij}, $V_n(\omega)\cdot V_n(\omega^{-1})$ 的 (i,j) 位元素为

$$\begin{aligned}u_{ij}&=\sum_{0\le k<n}\omega^{ik}(\omega^{-1})^{kj}\\&=\sum_{0\le k<n}(\omega^{i-j})^k\\&=n\delta_{ij}.\end{aligned}$$

因此 $V_n(\omega)\cdot V_n(\omega^{-1}) = n(\delta_{ij}) = nI_n$, 即 $(V_n(\omega))^{-1} = \frac{1}{n}V_n(\omega^{-1})$, 亦即 $(\mathrm{DFT}_\omega)^{-1} = \frac{1}{n}\mathrm{DFT}_{\omega^{-1}}$. □

注15. 这样一来, 系数表示和点值表示之间的正逆转换可以用相同的办法去处理, 也就是说可以采用统一的方法来计算单位原根处的多点求值和插值, 这是适当地选取插值点带来的好处之一, 现在 f 和 g 的循环卷积可以表示成

$$f * g = \frac{1}{n}\mathrm{DFT}_{\omega^{-1}}(\mathrm{DFT}_\omega(f)\cdot \mathrm{DFT}_\omega(g)),$$

其中 $\cdot$ 表示一维向量之间逐点相乘.

FFT

如果采用普通的多项式多点求值方法, 譬如说 Horner 法则, 计算 n 阶 DFT 大约需要 n^2 次系数乘法. 为了寻找更快的方法, 需要对 DFT 的结构进行仔细分析.

先来看向量 $\mathrm{DFT}_\omega(f)$ 的第 j 个分量 $f(\omega^j) = \sum\limits_{k=0}^{n-1} f_k\omega^{jk}$. 因为 ω 是 n 次单位原根, 所以 ω^{jk} 至多只有 n 个不同的值 $\omega^0, \omega^1, \ldots, \omega^{n-1}$, 特别地, 如果 j 是 n 的因子, 那么 ω^j 是 $\frac{n}{j}$ 次单位根, 因此 $(\omega^j)^k$ 至多只有 $\frac{n}{j}$ 个不同的值. 如果我们已经知道了哪些 ω^{jk} 具有相同的值, 就可以像合并同类项一样将对应于相同的 ω^{jk} 的系数 f_k 先加在一起, 然后再和 ω^{jk} 相乘, 这样就可以有效地减少乘法的次数, 跟计算 $ab + ac$ 时转而计算 $a(b + c)$ 是一个道理.

设 $0 \le k_1 < k_2 < n$, 那么

$$\omega^{jk_1} = \omega^{jk_2} \Leftrightarrow jk_1 \equiv jk_2 \mod n,$$

即 $n|j(k_1 - k_2)$, 一般地, n 是素数的整数次幂, 不妨设 $n = p^m$, 其中 p 为素数, 设 j 的素因子分解中 p 的幂次为 $m' < m$, 那么应该有 $p^{m-m'}|k_1 - k_2$, 这可以当作合并 ω^{jk} 的系数时的依据. 广义的 FFT 可以处理 p 为任意素数的情形, 我们这里接下来要讨论的的 FFT 则特指基 $p = 2$ 的 FFT.

为了高效地多点求值, FFT 方法运用了分治策略. 设 $n = 2^m$ 次多项式 f 的系数表示向量为 $f = (f_0, f_1, \ldots, f_{n-1})$, 将它依下标的奇偶性分成两组得到

$$f^{[0]} = (f_0, f_2, \ldots, f_{n-2}),$$
$$f^{[1]} = (f_1, f_3, \ldots, f_{n-1}),$$

那么向量 $\mathrm{DFT}_\omega(f)$ 的第 j 个分量

$$f(\omega^j) = \sum_{k=0}^{n-1} f_k\omega^{jk} = \sum_{k=0}^{n/2-1} f_{2k}(\omega^{2j})^k + \omega^j \sum_{k=0}^{n/2-1} f_{2k+1}(\omega^{2j})^k,$$

注意到 $\omega^{j+n/2}=-\omega^j, \omega^{j+n}=\omega^j$, 因此可以改写成

$$\begin{aligned}\mathrm{DFT}_\omega(f)[j] &= \mathrm{DFT}_{\omega^2}(f^{[0]})[j]+\omega^j\mathrm{DFT}_{\omega^2}(f^{[1]})[j],\\ \mathrm{DFT}_\omega(f)[j+n/2] &= \mathrm{DFT}_{\omega^2}(f^{[0]})[j]-\omega^j\mathrm{DFT}_{\omega^2}(f^{[1]})[j],\end{aligned}$$

其中 $0\le j<n/2$. 如果从多项式的角度来看, 这其实就是 $f(x)=f^{[0]}(x^2)+xf^{[1]}(x^2)$.

经过以上的分析, 现在可以写出 FFT 的详细过程了.

算法1.6 (FFT).

输入:$n-1$ 次多项式的系数表示向量 $f=(f_0,f_1,\ldots,f_{n-1})$, n 次单位原根 ω_n, 其中 $n=2^m$.

输出:$\mathrm{DFT}_{\omega_n}(f)$.

1. 若 n 等于 1, 返回 f.
2. 令 $f^{[0]}=(f_0,f_2,\ldots,f_{n-2})$, $f^{[1]}=(f_1,f_3,\ldots,f_{n-1})$, $\omega=1$.
3. 计算 $y^{[0]}=\mathrm{DFT}_{\omega_n^2}(f^{[0]})$, $y^{[1]}=\mathrm{DFT}_{\omega_n^2}(f^{[1]})$.
4. k 从 0 到 $n/2-1$, 顺次计算 $t=\omega y_k^{[1]}$, $y_k=y_k^{[0]}+t$, $y_{k+n/2}=y_k^{[0]}-t$, $\omega=\omega\omega_n$.
5. 返回 y.

设 $T(n)$ 是利用 FFT 进行多点求值时所需要的系数乘法的次数, 那么 $T(n)=O(n\log n)$, 如果环 R 为复数域, 那么利用 FFT 进行整数乘法时需要考虑浮点运算的截断误差和截断位数, 这种乘法算法的复杂度为 $O(n\log^* n)$, 其中 $\log^* n=(\log n)(\log\log n)\cdots$.

采用 FFT 计算 DFT 比直接按照定义计算要快得多, 它成功的主要原因是利用了单位原根 ω 的特殊性质和分治策略. FFT 在信号处理, 多媒体数据压缩等领域被广泛使用, 人们对基本算法进行了许多改进以获得更高的速度和适应于特定的硬件. 关于 FFT 更详细的介绍请参阅 [60], [174] 和 [104].

有限域上的 FFT

如果 $R=\mathbb{C}$, 即使 m 很大, 也很容易求出 $n=2^m$ 次单位原根. 但是对于有限域, 前面已经提到过, 如果 q 是素数的整数次幂, 那么当且仅当 $n|q-1$ 时有限域 $\mathbb{F}_q$ 才存在 n 次单位原根, 当 m 很大时, 并非总能轻易地找到这样的素数 q.

尽管如此, 相比于复数域而言, 使用有限域上的原根做 FFT 不会涉及到浮点操作, 因此也不用考虑中间精度的问题, 正因为这样, 在利用 FFT 计算整系数多项式的乘积或者计算高精度整数乘法时, 我们还是希望使用有限域. 注意到有限域 $F_q = \mathbb{Z}/q\mathbb{Z}$ 上的算术都是以 q 为模的, 因此利用 F_q 上的 FFT 求出的点值向量的各项系数都不应该超过 q, 否则最后无法将结果还原到 $\mathbb{Z}$ 中.

设 $a = (a_s \dots a_0)_B$, $b = (b_s \dots b_0)_B$, 那么 a, b 分别对应于多项式

$$a(B) = \sum_{k=0}^{s} a_k B^k, \quad b(B) = \sum_{k=0}^{s} b_k B^k,$$

a, b 的乘积 c 对应于多项式 $C(B) = \sum\limits_{k=0}^{2s} c_k B^k$, 其中

$$c_k = \sum_{i+j=k} a_i b_j < \sum_{i+j=k} B^2 \le (s+1) \cdot B^2, \quad 0 \le k \le 2s.$$

在典型的计算机代数系统中, 机器精度整数的溢出界 $B = 2^{64}$, 整数位数 s 的界也取为 2^{64}, 那么根据 F_q 的限制应该有 $q > (s+1)B^2 \approx 2^{192}$, 这样大的素数 q 以及有限域 F_q 是不容易使用的.

为了解决这个问题, 可以借用模方法中的技术, 选取三个机器精度的素数 $2^{63} \le q_0, q_1, q_2 < 2^{64}$, 分别在 F_{q_i} 中计算出模乘积 $a \cdot b \bmod q_i$, 然后利用中国剩余定理(参见 [6])计算出 $\mathbb{Z}$ 中的乘积 $a \cdot b$.

考虑到中国剩余定理的计算, 利用有限域上的 FFT进行整数乘法的计算复杂度为

$$O(n(\log n)(\log\log n)).$$

关于有限域上的 FFT 更详细的介绍请参阅 [174] 和 [154].

除了以上两种最常用的算法之外, 还有许多类似于 FFT 的算法也可以用于快速计算 DFT, 如 NTT(Number theoretic transform), WFT(Winograd Fourier Transform)和 PFT(Polynomial Fourier Transform)等, 关于 FFT, NTT, WFT 和 PFT 更详细的介绍请参阅 Nussbaumer[85] 或中译本 [86].

2 素数判定

素数判定(Primality Test)是一个数论中十分基本, 却又趣味盎然的问题. 判定一个整数是否是素数, 最为朴素的想法是直接利用素数的定义, 用小的素数去一一试除, 如果能整除的话, 那就能确定无疑为合数了. 根据 Mertens 定理(参见 [119]), 可以估算出大约有 76%的奇数有小于 100 的素因子, 可见这种最平凡的方法有时十分有效. 在本章中, 我们将用 N 来表示待判定的目标数.

相比整数的因子分解, 素数判定一直以来被认为是较为容易的问题. 2002 年三位印度计算机科学家 Agrawal, Kayal, Saxena[19] 找到了素数判定的多项式算法(被称为 AKS 算法), 其时间复杂度为 $O(\ln^{12} N)$. AKS 算法在理论上有重要的意义, 不过实践中还要更多地考虑效率问题. 实践中的素数检测方法大致分为两类, 一类是确定性的, 例如 Lehmer $N-1$ 检测, Lucas $N+1$ 检测, 椭圆曲线素性证明(ECPP)等等, 当输出结果为“素数”时, 能够保证被检测数一定为素数; 另一类是概率性的, 如 Rabin-Miller 检测, Baillie-PSW 检测等等, 当输出结果为“素数”时, 仅以一定的高概率保证被检测数的素性. 不过概率性检测一般要比确定性检测快得多.

APRCL 方法将 Fermat 类型的想法运用到分圆域中, 最先由 Adleman, Pomerance, Rumely[18] 提出, 后经 Cohen, Lenstra[57] 的改进, 时间复杂度为“近似”多项式的 $O((\ln N)^{c \ln\ln\ln N})$, 其中 c 为一个常数. 椭圆曲线素数证明最早由 Goldwasser, Kilian[75] 提出, 后经 Atikin, Morain[20] 改进和实现, 平均时间复杂度达到了 $O(\ln^6 N)$. 这两种方法已经成为目前实践上最快的确定性检测方法, 并在密码学等领域有重要的应用. 鉴于我们的目的, 不能用太多篇幅来介绍这两种方

法, 详细可参阅文献 [56].

有些素数判定方法严格来说应当是合性检测(Compositeness Test), 这类方法总可以有效地把素数判定出来, 而有可能把一个合数判定为素数. 也就是说, 此类方法判定一个数为合数总是准确无误的, 而“漏网的”, 即无法判定的合数往往称为伪素数(Pseudoprimes).

阅读本章需要读者初等数论和抽象代数的基础知识(例如 [10], [6]). 本章的主要参考书是 [146], [56] 和 [8]. 读者也可以参考写的十分通俗易懂的 [15] 及其他算法/计算数论方面的相关文献.

2.1 Fermat 检测

几乎所有素数检测的想法之基石均为著名的 Fermat 小定理, 即若 p 为素数, $(a,p)=1$, 则

$$a^{p-1}\equiv 1 \pmod{p}.$$

由此我们得到最简单的检测算法 — Fermat 合性检测.

算法2.1 (Fermat 小定理作为合性检测).

任取整数 a, 如果 $(a,N)=1$ 且 $a^{N-1}\not\equiv 1 \pmod{N}$, 则输出 N 为合数, 否则输出 N 可能为素数.

定义2.1 (Fermat 伪素数). 满足 $a^{N-1}\equiv 1 \pmod{N}$ 的合数 N 称为一个(对于基 a)的 Fermat 伪素数.

在 25×10^9 以下, 有 21853 个对于基 $a=2$ 的 Fermat 伪素数(参考 [144]), 如果对其再进行基为 $a=3$ 的检测的话, 伪素数将还剩下 4709 个, 对于 $a=2,3,5$ 有 2552 个, $a=2,3,5,7$ 有 1770 个.

运用 Fermat 检测的关键是如何快速计算 $a^d \mod N$. 求幂运算采取二进算法 4.1 , 4.2 , 可以使 Fermat 检测的算法复杂度降到平均 $1.5\log N\cdot M(N)$(其中 $M(N)$ 表示乘法运算的复杂度), 已经为多项式阶的算法了. 使用 Montgomery 约化算法 4.3 可以更进一步地提高速度.

Camichael 数是看起来比 Fermat 伪素数条件更“苛刻”的一类数.

定义2.2 (Camichael 数). 如果合数 N, 对任意满足 $(a,N)=1$ 的整数 a 都有 $a^{N-1}\equiv 1 \pmod{N}$, 则称 N 为 Camichael 数.

也就是说, Camichael 数都将会是 Fermat 检测的“漏网之鱼”. 最小的 Camichael 数的例子是 $561 = 3 \cdot 11 \cdot 17$. 在 25×10^9 以下一共有 2163 个 Camichael 数, 它比对基 $a = 2, 3, 5, 7$ 的 Fermat 伪素数还要来得多的原因是, 如果恰巧 N 有 2, 3, 5, 7 的因子, N 便可以是 Camichael 数而非 Fermat 伪素数.

2.2 Euler 检测

Euler 检测基于 Euler 的二次剩余定理, 即若 p 为奇素数, $(a, p) = 1$, 则

$$a^{(p-1)/2} \equiv \left(\frac{a}{p}\right) \pmod{p}.$$

其中 $\left(\frac{a}{p}\right)$ 是 Legendre 符号.

算法2.2 (Euler 判则作为合性检测)**.**
设 N 为奇数, 任取整数 a, 满足 $(a, N) = 1$:

1. 如果 $a^{(N-1)/2} \not\equiv \pm 1 \pmod{N}$, 则输出 N 为合数.
2. 如果 $a^{(N-1)/2} \equiv \pm 1 \pmod{N}$ 且 $a^{(N-1)/2} \not\equiv \left(\frac{a}{N}\right) \pmod{N}$, 则输出 N 为合数.
3. 如果 $a^{(N-1)/2} \equiv \left(\frac{a}{N}\right) \pmod{N}$, 则输出 N 可能为素数.

定义2.3 (Euler 伪素数)**.** 如果合数 N, $(a, N) = 1$, 满足 $a^{(N-1)/2} \equiv \left(\frac{a}{N}\right) \pmod{N}$, 则称 N 为(对于基 a 的)Euler 伪素数.

定义2.4 (强伪素数)**.** 设 N 奇合数, $N - 1 = d \cdot 2^s$, d 为奇数. N 被称为(对于基 a 的)强伪素数, 如果

$$a^d \equiv 1 \pmod{N}$$

或者对于某一个 $r (0 \le r \le s - 1)$ 有

$$a^{d \cdot 2^r} \equiv -1 \pmod{N}.$$

注16. 强伪素数的定义可以这样看出: 若 N 为素数, 则

$$\begin{aligned} a^{N-1} - 1 &= (a^d - 1)(a^d + 1)(a^{2d} + 1)(a^{4d} + 1) \cdots (a^{2^{s-1}d} + 1) \\ &\equiv 0 \pmod{N}. \end{aligned}$$

对于 Euler 伪素数和强伪素数均不会出现类似 Fermat 伪素数的情况, 也就是说, 只要测试的基 a 足够多, 最终总可以将合数和素数分辩开来. 25×10^9 以下只有 13 个同时为对基 2, 3, 5 的强伪素数.

注17. [144] 证明了, Euler 伪素数必定是强伪素数.

2.3 Lehmer $N-1$ 型检测

有时候 N 的素性很难判定, 而 $N\pm1$ 的分解却很容易得到. 这一节的方法便是依赖于 $N-1$ 的素因子分解.

定理2.1 (Fermat 小定理逆定理, Lehmer). *设有素因子分解 $N-1=q_1^{\beta_1}\cdots q_n^{\beta_n}$. 如果对 a 有*

$$a^{(N-1)/q_j}\not\equiv 1 \pmod N, \qquad \forall j=1,2,\dots,n,$$
$$a^{N-1}\equiv 1 \pmod N. \tag{2.1}$$

则 N 为素数.

证明. 从简化剩余系的乘法群 $M_N=(\mathbb{Z}/N\mathbb{Z})^*$ 角度来看, 条件保证了 a 为 M_N 的一个 $N-1$ 阶元素, 而 $|M_N|=\varphi(N)$(φ 为 Euler 函数), 从而有 $\varphi(N)\geq N-1$, 这就保证了 N 为素数. □

注18. 这里不要 $(a,N)=1$ 的条件, 是因为其已经被包括在式 (2.1) 之中了.

寻找 M_N 的生成元(甚至只寻找模 N 的一个二次剩余)都是非常困难的事情, 目前还没有有效的确定性的方法. 一个现实的问题是如果 N 为伪素数, 如何才能快速的找到 Lehmer $N-1$ 检测中符合条件的 a, 即 M_N 的一个生成元呢? 有一些步骤可以采用:

1. 排除所有 $\left(\frac{a}{N}\right)=1$ 的 a, 这就排除了一半可能的 a!
2. 从较小的 q_j 开始检测, 因为 q_j 越小, 失败的可能性越大.

定理2.2 (Selfridge 的多基推广). *如果 $\forall q_j$, $\exists a_j$ 使得*

$$a_j^{(N-1)/q_j}\not\equiv 1 \pmod N$$

且

$$a_j^{N-1}\equiv 1 \pmod N.$$

则 N 为素数.

证明. 设 a_j 模 N 的阶为 r_j, 则以上两式表明 $r_j \nmid \frac{N-1}{q_j}$ 且 $r_j \mid N-1$, 从而 $q_j^{\beta_j} \mid r_j \Rightarrow N-1 \mid \varphi(N) \Rightarrow \varphi(N) = N-1$. □

注19. 这个方法对 M_N 的生成元很大的时候非常有用, 因为我们不必找出一个"共同"的 a 来.

定理2.3 (Pepin 定理). *设 $F_n = 2^{2^n}+1$ 为 Fermat 素数$(n \geq 1)$. 则 F_n 为素数当且仅当*

$$3^{2^{2^n-1}} \equiv -1 \pmod{F_n}$$

证明. 用定理 2.1 即可, 只需注意到 3 必为 $12n \pm 5$ 型素数的二次非剩余. □

注20. 使用这个方法, 在 1963 年 [159] 证明了 F_{14}(有 4933 个十进制位)是合数, 尽管但现在为止还不知道其任何一个因子.

下面的定理让我们可以拥有不必完全分解 $N-1$ 的便利.

定理2.4 (Lehmer 定理的放宽版本). *设 $N-1 = RF$, $(R,F)=1$ 且 $R < F$, 有素因子分解 $F = \prod_{j=1}^{n} q_j^{\beta_j}$. 若有 a 使得*

$$a^{(N-1)/q_j} \not\equiv 1 \pmod{N}, \qquad \forall j = 1,2,\ldots,n$$

且

$$a^{N-1} \equiv 1 \pmod{N},$$

则 N 为素数.

证明. 设 p 为素数, $p \mid N$, 设 a 模 p 的阶是 d, 则由条件有 $d \mid N-1$, 但 $d \nmid \frac{N-1}{q_j}$. 从而 $\forall j$, $d \mid \prod_{i=1}^{n} q_i$, 但 $d \nmid R \cdot q_j^{\beta_j-1} \prod_{i\neq j} q_i^{\beta_i}$. 故有

$$q_j^{\beta_j} \mid d \Rightarrow F \mid d \Rightarrow F \mid p-1 \Rightarrow F \leq \sqrt{N}.$$

矛盾! □

定理2.5 (Proth 定理). *设 $N = h \cdot 2^n + 1$, 其中 h 为奇数, $2^n > h$. 如果存在 a 使得*

$$a^{(N-1)/2} \equiv -1 \pmod{N}.$$

则 N 为素数.

证明. 在定理 2.4 中令 $F = 2^n$ 即得. □

2.4 Lucas 伪素数检测与 $N+1$ 型检测

如果 $N-1$ 也是很难分解的, 这里的方法就将分解的困难转移到 $N+1$ 上去. 想法是与 $N-1$ 型检测是类似的, 不过要将 Fermat 小定理从 $\mathbb{Z}$ 上推广到二次域的代数整数环上去, 替代那里的 $a^{N-1}-1$ 的则是所谓 Lucas 序列 U_{N+1}.

设 D 为无平方整数, 记 O 为二次域 $\mathbb{Q}(\sqrt{D})$ 的代数整数环, 我们熟知(参见 [6], P139)当 $D \equiv 2,3 \pmod 4$ 时,

$$O = \{m + n\sqrt{D} \mid m, n \in \mathbb{Z}\},$$

而当 $D \equiv 1 \pmod 4$ 时,

$$O = \left\{ \frac{m + n\sqrt{D}}{2} \,\middle|\, m, n \in \mathbb{Z}, m \equiv n \pmod 2 \right\}.$$

在 O 中可以很容易地讨论同余的概念, 称 $a \equiv b \pmod M$, 若 $a-b$ 属于 M 在 O 中生成的理想. 记 $\bar{a}$ 为 a 在 O 中的共轭, 即 $\overline{r+s\sqrt{D}} = r - s\sqrt{D}$, 则在 O 中我们有如下 Fermat 小定理的类比.

定理2.6 (二次域中的 Fermat 小定理)**.** *设 $a \in O$, p 为奇素数, $p \nmid D$, 则有*

$$a^p \equiv \begin{cases} a & 若 \left(\dfrac{D}{p}\right) = 1 \\ \bar{a} & 若 \left(\dfrac{D}{p}\right) = -1 \end{cases} \pmod p.$$

证明. 设 $2a = r + s\sqrt{D}$, $r, s \in \mathbb{Z}$, 则由 $\mathbb{Z}$ 上的 Fermat 小定理, 二项展开及 Euler 二次剩余定理得

$$\begin{aligned} 2a^p &\equiv (2a)^p \equiv (r + s\sqrt{D})^p \equiv r^p + (s\sqrt{D})^p \\ &\equiv r + sD^{\frac{p-1}{2}}\sqrt{D} \\ &\equiv r + s\left(\frac{D}{p}\right)\sqrt{D} \pmod p. \end{aligned}$$

从而得到所要结论. □

定理 2.6 中的乘幂不容易计算, 为了有效地利用定理 2.6 的结论, 我们引入 Lucas 序列的概念, Lucas 序列实际上是一类简单的二阶递推序列.

定义2.5 (Lucas 序列)**.** 设 $P, Q \in \mathbb{Z}$, 特征方程 $\lambda^2 - P\lambda + Q = 0$ 的两根为 a, b. 则 P, Q 对应的 Lucas 序列定义为

$$U_n = \frac{a^n - b^n}{a - b}, \qquad V_n = a^n + b^n.$$

注21. 由特征方程对应的递推关系, 知道 $\{U_n\},\{V_n\}$ 均为 $\mathbb{Z}$ 中的序列.

引理2.1. 记号同定义 2.5 , 设整数 M 满足 $(QD,M)=1$, 则 $a,b,a-b$ 均模 M 可逆. 并且 $U_k\equiv 0 \pmod M$ 当且仅当 $(ab^{-1})^k\equiv 1 \pmod M$.

证明. 由 $(Q,M)=1$ 知 Q 模 M 可逆, 而 $ab=Q$, 知 $ab\cdot Q^{-1}\equiv 1 \pmod M$, 从而 a,b 均模 M 可逆. 又 $a-b=\sqrt{D}$, 而 $D^{\varphi(M)}\equiv 1 \pmod M$, 可知 $(a-b)^{2\varphi(N)}\equiv 1 \pmod M$, 从而 $a-b$ 也模 M 可逆.

由于 $a-b$ 模 M 可逆, $U_k\equiv (a-b)^{-1}(a^k-b^k) \pmod M$, 从而

$$U_k\equiv 0 \pmod M \Longleftrightarrow a^k\equiv b^k \pmod M \Longleftrightarrow (ab^{-1})^k\equiv 1 \pmod M.$$

引理得证. □

定理2.7. 设 p 为素数, 满足 $(2QD,p)=1$, 记 $\varepsilon=\left(\frac{D}{p}\right)$, 则有 $U_{p-\varepsilon}\equiv 0 \pmod p$.

证明. 若 $\left(\frac{D}{p}\right)=1$, 则由定理 2.6 可知 $a^{p-1}\equiv b^{p-1}\equiv 1 \pmod p$, 从而 $(ab^{-1})^{p-1}\equiv 1 \pmod p$, 由引理 2.1 知 $U_{p-1}\equiv 0 \pmod p$.

若 $\left(\frac{D}{p}\right)=-1$, 则由定理 2.6 可知 $a^{p+1}\equiv a\bar{a}\equiv ab \pmod p$, 同理 $b^{p+1}\equiv ba \pmod p$, 从而 $(ab^{-1})^{p+1}\equiv 1 \pmod p$, 由引理 2.1 知 $U_{p+1}\equiv 0 \pmod p$, 这就完成了证明. □

由上面的定理我们自然地得到 Lucas 伪素数的概念.

定义2.6 (Lucas 伪素数)**.** 设 N 为奇合数, 若存在 Lucas 序列 $\{U_n\}$ 使得 $\varepsilon=\left(\frac{D}{N}\right)\neq 0$ 且 $U_{N-\varepsilon}\equiv 0 \pmod N$, 则称 N 为 Lucas 伪素数.

正如我们在 Fermat 检测和 Euler 检测中看到的, 一个伪素数概念对应着一个合性检测算法, 由此我们立即可以得到 Lucas 伪素数检测算法.

算法2.3 (Lucas 伪素数检测)**.**

选取 Lucas 序列参数 P,Q, 使得 $\varepsilon=\left(\frac{D}{N}\right)\neq 0$. 若 $U_{N-\varepsilon}\not\equiv 0 \pmod N$ 则输出 N 为合数, 否则输出 N 可能为素数.

注22. 由于来源的不同, 我们可以期望一个数同时为 Lucas 伪素数和 Fermat 伪素数(或强伪素数)的可能情形不大, 这样的观察也被用于 Baillie-PSW 检测算法 2.6 中.

我们已经得到了 Lucas 伪素数合性检测, 和 $N-1$ 型检测类似, 我们还能够进行素性的确证. 在这里, 替代 $N-1$ 的分解的是 $N+1$ 的分解, 我们可以看到下面的定理与定理 2.1 是多么的“形似”.

定理2.8 (Lucas 检测). *设有素因子分解 $N+1=\prod_{j=1}^{n} q_j^{\beta_j}$, 若有 Lucas 序列 $\{U_n\}$, 使 $(2QD,N)=1$, 满足*

$$(U_{(N+1)/q_j},N)=1, \qquad \forall j=1,2,\ldots,n$$

且

$$U_{N+1}\equiv 0 \pmod N,$$

则 N 为素数.

证明. 设 p 为 N 的一个素因子, 设 k 为最小的正整数使得 $(ab^{-1})^k\equiv 1 \pmod p$. 由条件知 $U_{N+1}\equiv 0 \pmod p$, $U_{(N+1)/q_j}\not\equiv 0 \pmod p$, 从而根据引理 2.1 知 $k\mid N+1$, $k\nmid (N+1)/q_j$, $\forall j$, 从而 $k=N+1$. 但根据 k 的最小性, 由定理 2.7 必有 $k\mid q+1$ 或 $k\mid q-1$. 从而 $q=N$, N 为素数. □

与 $N-1$ 型检测完全类似还可以得到以下结果.

定理2.9 (Lucas 检测的放宽版本). *设 $N+1=RF$, $(R,F)=1$ 且 $R<F$, 有素因子分解 $F=\prod_{j=1}^{n} q_j^{\beta_j}$. 若有 $\{U_k\}$ 为 Lucas 序列, $(2QD,N)=1$, 满足*

$$(U_{(N+1)/q_j},N)=1, \qquad \forall j=1,\ldots,n$$

且

$$U_{N+1}\equiv 0 \pmod N.$$

则 N 为素数.

注23. Lucas 检测的一个优势在于 Lucas 序列可以通过递推快速地计算. 设

$$A_k=\begin{bmatrix} U_{k+1} & V_{k+1} \\ U_k & V_k \end{bmatrix},$$

则有

$$A_k=\begin{bmatrix} P & -Q \\ 1 & 0 \end{bmatrix} A_{k-1}=\cdots=\begin{bmatrix} P & -Q \\ 1 & 0 \end{bmatrix}^k A_0.$$

可以使用快速求幂的方法在 $O(\log k)$ 步中完成序列的计算.

注24. 上述基于 $N+1$ 分解的算法对 Mersenne 素数2^n-1 尤其简单, 因此也被著名的GIMPS[1](Great Internet Mersenne Prime Search)所采用. 目前为止已知的最大的 Mersenne 素数是 GIMPS 计划在 2008 年 8 月 23 日找到的第 45 个 Mersenne 素数, 共有 12978189 个十进制位. 有趣的是, 2009 年 4 月 12 日找到的第 47 个 Mersenne 素数要比第 45 个来的小.

定理2.10 (对 Mersenne 素数的 Lucas-Lehmer 检测)**.** *设 n 为奇数, $v_0=4$, $v_k=v_{k-1}^2-2$. 则 $M_n=2^n-1$ 为素数当且仅当*

$$v_{n-2}\equiv 0 \pmod{M_n}.$$

证明. 证明主要部分来源于定理 2.8 , 然而仍有一些技术上的区别, 这里就不赘述了. 完整的可见 Lehmer 在 1930 的证明 [112]. 另外 J. W. Bruce 在 1993 年给出了一个短至 2 页的“trival”版证明 [47]. □

2.5 概率性的检测方法

更加富有趣味的是素性的概率性检测方法. Knuth[104] 这样评价概率性的算法: “与其说算法重复地猜测错, 倒不如说由于硬件的失灵或宇宙射线的原因, 我们的计算机在它的计算中丢了一位. ”概率性的算法使我们对传统的可靠性产生疑问: 我们是否真的需要“素性”的严格确证? 概率性算法最早由 Solovay, Strassen[163] 于 1974 年提出, Rabin-Miller 的改进方法为 Mathematica 软件所采用. Baillie-PSW 检测则综合了各种概率性检测方法, 目前为止仍没有找到失败的反例, 并且已经验证检测对于 10^{15} 以下的整数均是正确的.

2.5.1 Solovay-Strassen 检测

算法2.4 (Solovay-Strassen 检测)**.**

1. 随机生成满足 $1\le a\le N$ 的整数 a;
2. 若 $(a,N)=1$ 且 $\left(\frac{a}{N}\right)\equiv a^{(N-1)/2} \pmod{N}$, 则输出 N 为素数, 否则输出 N 可能为合数.

[1] http://www.mersenne.org/

定理2.11. *算法 2.4 重复执行 k 次, 给出错误的概率为 0(当 N 为素数时)或 2^{-k}(当 N 为合数时)*

证明. 只需证明当 N 为合数, $(a,N)=1$ 时, $\left(\frac{a}{N}\right) \equiv a^{(N-1)/2} \pmod N$ 的概率不超过 $\frac{1}{2}$ 即可. 设

$$G = \left\{ a + N\mathbb{Z} \;\middle|\; a \in \mathbb{Z}, (a,N)=1, a^{(N-1)/2} \equiv \left(\frac{a}{N}\right) \pmod N \right\}.$$

则 $G < M_N$, 因此只需证明 $G \neq M_N$ 即可.

如果 N 能分解为两个互素的非平凡因子 r, s, 我们证明必有 $\forall a$ 满足 $(a,N)=1$ 都有 $a^{(N-1)/2} \equiv 1 \equiv \left(\frac{a}{N}\right) \pmod N$(由 Jacobi 符号的性质便知这是不可能的). 若不然, 由中国剩余定理找到 b 满足 $b \equiv 1 \pmod r$ 且 $b \equiv a \pmod s$. 则有 $b^{(N-1)/2} \equiv 1 \pmod r$ 且 $b^{(N-1)/2} \equiv -1 \pmod s$, 而这与 $b^{(N-1)/2} \equiv \pm 1 \pmod N$ 是矛盾的!

如果 $N = p^e$ 为素数幂, 则 $\forall a$ 满足 $(a,N)=1$ 都有 $a^{p^e-1} \equiv 1 \pmod{p^e}$, 再由 Euler 定理知道 $p^{e-1}(p-1) \mid p^e - 1$. 而这也是不可能的. □

2.5.2 Rabin-Miller 检测

Solovay-Strassen 检测利用了 Euler 伪素数"并不多"的性质, 使用随机给出的基 a 多次重复进而给出高成功率的素性检测结果. 同样的想法也可以施用于强伪素数与 Lucas 伪素数, 前者便是著名的 Rabin-Miller 检测, 后者则被运用到 Baillie-PSW 检测中.

Rabin-Miller 检测可以看成我们在注 16 中的想法的一个具体化.

算法2.5 (Rabin-Miller 强伪素数检测).

设 $N - 1 = d \cdot 2^s$,

1. 随机生成满足 $1 \leq a \leq N$ 的整数 a;
2. 顺次计算
$$a^d, a^{2d}, a^{4d}, \ldots, a^{2^s \cdot d}.$$
若计算到第 k 步时有
   (a) $k = 1$ 且 $a^d \equiv 1 \pmod N$, 输出 N 可能为素数;
   (b) $k > 1$ 且 $a^{2^{k-1} \cdot d} \equiv -1 \pmod N$, 输出 N 可能为素数;
   (c) $k > s$, 输出 N 为合数.

注25. Rabin[145] 证明了算法 2.5 重复 k 遍, 给出正确判断的概率大于 $1-4^{-k}$.

注26. 使用单个基 a 的 Rabin-Miller 检测可能会遗漏许多基 a 下的强伪素数, 因此实践中的确需要使用多基的 Rabin-Miller 测试. 例如使用前 7 个素数组成的多基 Rabin-Miller 测试, 可以保证算法在 $341550071728321 \approx 3.4 \times 10^{14}$ 以下均是有效的, 而且这个数在用前 8 个素数为基的 Rabin-Miller 测试下没有改进.

2.5.3 Baillie-PSW 检测

只是使用多基的 Rabin-Miller 测试可能得不到好的效率. Baillie[23] 将基为 2 的 Rabin-Miller 检测与 Lucas 伪素数检测结合起来, 得到更为有效的素数检测方法. Pomerance, Selfridge, Wagstaff 在 [144] 中对小于 25×10^9 的数验证了算法的正确性. 尽管 Pomerance[143] 指出对于充分大的 N, 必定有 Baillie-PSW 检测的反例, 并且以 30 美元悬赏找出一个算法失败的反例(后来增加到 620 美元), 不过反例至今没有找到, 并且有经验估计认为第一个反例如果被找到的话, 至少得有 10000 个十进位长度(参见 [120]).

下面我们正式给出 Baillie-PSW 检测.

算法2.6 (Baillie-PSW 检测).

1. 使用小素数试除 N, 若能整除, 输出 N 为合数.
2. 使用基为 2 的 Rabin-Miller 强伪素数检测 2.5 , 若 N 非强伪素数, 输出 N 为合数.
3. 选择以下两种方法之一确定 Lucas 序列的参数 P, Q:
   - (Selfridge 建议)取 D 为 $5, -7, 9, -11, 13, \ldots$ 中使得 $\left(\frac{D}{N}\right) = -1$ 的第一个数, 选择 $P = 1$, $Q = \frac{1-D}{4}$;
   - (Baillie 建议)取 D 为 $5, 9, 13, 17, 21, \ldots$ 中使得 $\left(\frac{D}{N}\right) = -1$ 的第一个数, 选择 P 为 $> \sqrt{D}$ 的最小奇数, $Q = \frac{P^2-D}{4}$.
4. 使用 P, Q 为参数的 Lucas 伪素数检测算法 2.3 , 若 N 非 Lucas 伪素数, 输出 N 为合数, 否则输出 N 可能为合数. 用算法 4.5 检测 N 是否为完全平方数.

注27. 由于 $D = P^2 - 4Q^2$, 只需尝试模 4 余 1 的数. Selfridge 建议方法中不使用 -3 是因为当 $D = -3$ 时有 $P = Q = 1$, 将会产生以 $\{1, 1, 0, -1, -1, 0\}$ 为循环节的

周期 Lucas 序列, 这将使得每个奇合数变成关于 P, Q 的 Lucas 伪素数, 并非我们所想要的.

注28. 若第三步中尝试过程中计算出前若干个 $\left(\frac{D}{N}\right)$ 均为 1, 则 N 很可能为完全平方数, 此时可用算法 4.5 对 N 进行平方检测以防止额外的计算, 若 N 不是平方数则继续寻找 D 的过程. 另外尝试过程中若计算得 $\left(\frac{D}{N}\right) = 0$, 则意味着 N 必为合数, 可直接终止计算.

注29. [23] 证明了, 在第三步中, 两种方法取到合适的 D 的平均尝试次数小于 2.

3 整数因子分解

相对于素数判定来说, 因子分解的实现就没办法达到那么快速了. 因子分解至今仍没有类似于素数判定的多项式算法, 这也成为了 RSA 公钥系统安全得以保障的基础. 鉴于这两个问题的难度相差较大, 在我们施行分解之前, 最好是预先知道目标整数的确不是一个素数, 否则很可能花费了很大力气只干了素数判定的活 — 杀鸡用牛刀了.

因子分解的方法分为一般方法和特殊方法两大类, 通常倾向于先针对数的特殊性(例如 $N = 2^n - 1$)使用特殊方法, 如果目标数的形式不那么特殊, 再尝试使用一般方法. 当然, 前者往往要比后者快上许多.

我们在本章中着重介绍因子分解的一般方法, 且总是用 N 表示待分解的目标数. 阅读本章需要读者具有初等数论和抽象代数的基础知识(例如 [10], [6]). 本章的主要参考书是 [56], [174]. 读者也可以参考写的十分通俗易懂的 [15] 及其他算法/计算数论方面的相关文献.

3.1 试除法

无论素数判定还是因子分解, 试除法(Trial Division)都是首先要进行的步骤. 在试除的策略上有两种不同的选择:

- 用足够大的空间来储存试除用的素数因子. 储存方法可以相当紧凑, 比如使用一个大整数, 此大整数二进制表达的第 $2k-1, 2k$ 位来代表 $6k \pm 1$ 是否为素数.

- 不耗费大量空间来储存所有需要的素因子, 这时需要一个快速生成素数的子程序, 或者干脆只用 2, 3 以及 $6k \pm 1$ 型的整数来作为试除因子.

很少能发生一个数没有小因子的情况, 例如根据 Mertens 定理(参见 [119]), 奇数中没有 x 以下因子的比例

$$P = \prod_{p \geq 3}^{x} (1 - \frac{1}{p}) \sim \frac{2e^{-\gamma}}{\ln x}, \quad x \to +\infty.$$

可以知道 76%的奇数都有小于 100 的素因子, 而没有小于 10^8 因子的奇数比例仅为 6.1%(可参看 [146]). 因此在大多数情况, 试除法的第二种选择已经足够, 实现也是最为简单的.

3.2 Euclid 算法

Euclid 算法用于因子分解也非常简单. 我们预先计算好小于 100 的素数之积

$$p_0 = \prod_{\substack{2 \leq p \leq 97 \\ p\text{为素数}}} p = 23055679639455184247531021473317560 70.$$

然后将 p_0 与目标数 N 进行 Euclid 算法, 最终得到 p_0 与 N 的最大公因子, 继续分解公因子就可以得到在 100 以下的因子分解了. 同样可以预先计算出 100 到 200, 200 到 300 的素数乘积 p_1, p_2 等等. 这本质上是试除法的一个实现, 当 N 非常大时, 必须借助高精度算术来进行 N 除以 p 的操作, 因此频繁的试除会十分耗时, 而 Euclid 方法可以施行很少次数, 再在机器精度上完成最终的分解, 提高效率.

3.3 Pollard $p-1$ 方法

Pollard $p-1$ 方法由 Pollard[140] 于 1974 年提出, 其基本想法是这样的: 设素数 $p \mid N$, 由 Fermat 小定理, 又有 $p \mid a^{p-1} - 1$, 因此 $(a^{p-1} - 1, N)$ 就可能是 N 的一个非平凡因子. 当然, 问题在于我们并不知道 p 是多少. 一个合理的假设是 $p-1$ 的因子都很小, 比如说, $p-1$ 所有素因子都包含在因子基 $FB = \{p_1, p_2, \ldots p_m\}$ 中, 我们来尝试着找到一个 $c = \prod_{i=1}^{m} p_i^{\alpha_i}$ 能够"覆盖"$p-1$, 即是说 $p-1 \mid c$, 从而 $a^{p-1} - 1 \mid a^c - 1$, 因此我们可以转而求 $(a^c - 1, N)$ 来获得所要的非平凡因子. 例如设素因子上限为 B, 便可以简单的取 $c = B!$ 或是最小公倍数 $\mathrm{LCM}\{1, 2, \ldots, B\}$.

下面给出 Pollard $p-1$ 方法的一个版本:

算法3.1 (Pollard $p-1$ 方法).

1. 设素因子搜索的上限为 B, 生成 B 以下的形如 $p_i^{\alpha_i}$ 数对应的素数因子之表 $\{p_i\}$, 其中 p_i 的幂次用 p_i 代表, 即: 2, 3, 2, 5, 7, 2, 3, 11, 13, 2...
2. 随机选择正整数 a, 顺次计算
$$b_1 = a$$
$$b_{i+1} \equiv b_i^{p_i} \pmod{N} \quad i = 1, 2, \ldots$$
3. 定期检查(例如每当 n 为 20 的倍数时)$(b_n - 1, N)$, 若 $(b_n - 1, N) > 1$, 则得到一个 N 的因子; 否则继续第 2 步中的递推计算.

注30. 由于越小素数在 $p-1$ 分解中出现的幂次可能越高, FB 中小素数(例如 2, 3)应当较多重复出现, 第 1 步中的生成方法便考虑到了这一点. (实际上最终计算了 $\mathrm{LCM}\{1, 2, \ldots, B\}$.)

注31. 在极少的情形, 也可能出现 $(b_n - 1, N) = N$, 即所有 N 的素因子都同时出现在了 $b_n - 1$ 之中, 这时可以重新选取定期检查的时机或者换一个 a 进行计算.

注32. 另一种类似的 Williams $p+1$ 方法依赖于 $p+1$ 只有小的因子, 著名的 Lucas 序列替代了这里的 a 的幂次, 乘法群 $\mathbb{Z}/p\mathbb{Z}(\sqrt{t})^*$($t$ 为 p 的二次非剩余)代替了乘法群 $\mathbb{Z}/p\mathbb{Z}^*$. 因此 Pollard $p-1$ 方法与 Williams $p+1$ 方法的关系就好像素数检测中的 Lehmer $N-1$ 型检测与 Lucas $N+1$ 型检测的关系一样. 具体可参看 [185].

注33. 实践中 B 一般取 10^6 左右.

注34. Pollard $p-1$ 方法的时间复杂度为 $O(N^{1/4+\varepsilon})$, 其中 ε 为一个正数(参见 [174]).

3.4 Pollard ρ 方法

目前几乎所有实用的分解方法都是概率性的算法, 目标是找到能计算 x 的算法, 使得 $(x, N) > 1$ 的概率较大(而最大公因子可以很快地计算). 上面的 Pollard $p-1$ 就是一例, 下面即将看到的 Pollard ρ 方法也不例外.

Pollard ρ 方法由 Pollard[141] 在 1975 年提出, 它来自一个有趣的事实: 随机选取大约 $c\sqrt{p}$ 个整数(c 为一个常数), 就有很大概率在这些整数中找到两个是模 p

同余的. 实践中可以采用同余递推序列

$$x_{i+1} \equiv f(x_i) \pmod{N}$$

来产生伪随机数, 其中 f 为映射: $\mathbb{Z}/N\mathbb{Z} \to \mathbb{Z}/N\mathbb{Z}$. 设 p 是 N 的一个因子, 且找到 $x_j \equiv x_i \pmod{p}$, 则计算 $(x_j - x_i, N)$ 便可能得到 N 的一个非平凡因子.

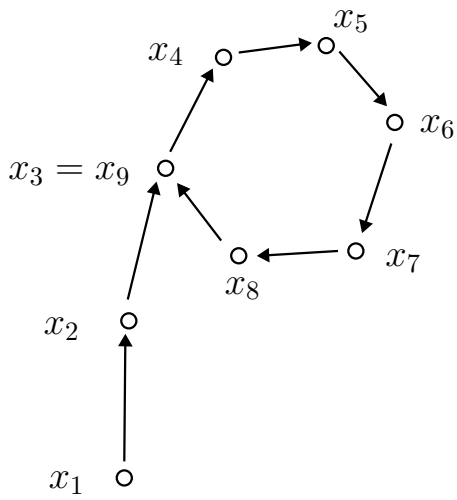

图 3.1: Pollard ρ 方法

由 $\mathbb{Z}/p\mathbb{Z}$ 的有限性, 如上定义的一阶的递推序列 $\{x_i\}$ 在模 p 意义下必定是最终循环的(如图, 看上去就像希腊字母 ρ). 设其开头的非循环部分长度为 m, 循环节长度为 l. 著名的 Floyd 算法可以在 $m + l$ 步内高效地找出序列中的两个重复元素, 并且只用常数的储存空间.

算法3.2 (Floyd)**.**

1. 依次判断是否 $x_{2i} = x_i$, $i = 1, 2, 3, \cdots$
2. 若相等, 则终止; 否则继续第 1 步.

算法的有效性. 只需证明必定存在 i 满足 $m < i \leq m + l$ 且 $x_{2i} = x_i$. 由于 $x_{2i} = x_i$ 等价于 $l \mid i$, 因此取 i 为 $(m, m+l]$ 中 l 的倍数即可. 算法过程中不必保存所有 $\{x_i\}$, 可以存下当前的 x_i, x_{2i} 并递推计算 $x_{i+1} = f(x_i)$, $x_{2(i+1)} = f(f(x_{2i}))$. □

实践中常采用的 f 为

$$x_{i+1} \equiv x_i^2 + a \pmod{N},$$

选择二次的递推序列一方面能提供足够的随机性, 另一方面计算起来也非常简便.

算法3.3 (Pollard ρ).

1. 随机选取 a 与 x_1.
2. 顺次计算 $x_{i+1} = x_i^2 + a \mod N$.
3. 计算
$$Q_n = \prod_{i=1}^{n}(x_{2i} - x_i) \mod N.$$
4. 每隔一段时间(例如 $k = 20$), 检测 (Q_{nk}, N), 若非平凡则算法终止, 否则继续第 2 步.

注35. 当 N 较大时, 对每个 i 都去检测 $(x_{2i} - x_i, N)$ 可能会耗费大量时间, 因为我们的目标只是得到 N 的非平凡因子, 可以通过计算 Q_n, 再定时检测 (Q_{nk}, N) 来减少计算次数.

注36. 如同 Pollard $p-1$ 方法, 也可能出现计算出来的最大公因子 $(Q_n, N) = N$, 这时可改变检测间隔 k 或干脆改变 a 重新进行计算.

注37. Pollard ρ 方法的时间复杂度为 $O(N^{1/4+\varepsilon})$(参见 [174]). 实际上复杂度依赖于 N 的最小素因子 $P^-(N)$, 在分离 N 的小因子时尤其有效.

注38. 1980 年, Brent[39] 给出了 Pollard ρ 方法的一个改进, 在分解整数时, 该方法平均能够加速 24%. 这个改进是针对 Floyd 的算法 3.2 的, 因为在 Floyd 算法中, 往往要重复计算 $x_2, x_4, x_6, \cdots$ 等, Brent 有如下改进, 无需重复计算, 但仍能同样有效地找出重复元素, 并且只要常数的储存空间.

算法3.4 (Brent 的改进).

1. 令 $j = 2$, $i = 1$, 若 $x_j = x_i$, 则算法终止.
2. 若 j 为 2 的幂, 即 $j = 2^k$, 令 $i = j$, 依次令 $j = 3\cdot 2^{k-1}+1, 3\cdot 2^{k-1}+2, \cdots, 2^{k+1}$, 判断是否有 $x_j = x_i$, 若相等则算法终止.

算法的有效性. 注意算法过程中 $j - i$ 能够依次遍历所有正整数, 重复算法 3.2 的论证可知. □

3.5 平方型分解(SQUFOF)

平方型分解(SQUare FOrm Factorization)是由 Shanks 在大约三十前发展的算法, 但他从来没有正式发表过(参见 [79]). 尽管 SQUFOF 复杂度为 $O(N^{1/4+\varepsilon})$ 也是一个指数级的算法(而下面介绍的 CFRAC, ECM, QS 等都是次指数级的), 但其仍有自身的优势: 一方面算法十分简洁优美, 便于实现(甚至可以在袖珍计算器上实现), 并且在 10^{10} 到 10^{18} 范围的整数分解仍然是最快的.

SQUFOF依赖于对二次域结构的分析, 我们在这里仅给出算法的描述, 略去证明, 具体可参见 [79]:

算法3.5 (SQUFOF).

设 N 非平方数, 非素数, 以下算法输出 N 的一个非平凡因子.

1. 设 $P_0 = \lfloor\sqrt{N}\rfloor$, $Q_0 = 1$, $Q_1 = N - P_0^2$.
2. 顺次计算 $b_i = \left\lfloor \frac{\lfloor\sqrt{N}\rfloor + P_{i-1}}{Q_i} \right\rfloor$, $P_i = b_i Q_i - P_{i-1}$, $Q_{i+1} = Q_{i-1} + b_i(P_{i-1} - P_i)$, 直到 Q_k 为完全平方数. $(i = 1, 2, \ldots)$
3. 计算 $b_0 = \left\lfloor \frac{\lfloor\sqrt{N}\rfloor - P_{i-1}}{\sqrt{Q_k}} \right\rfloor$, $P_0 = b_o\sqrt{Q_k} + P_{i-1}$, $Q_0 = \sqrt{Q_k}$, $Q_1 = \frac{N - P_0^2}{Q_0}$.
4. 重复第二步中的计算, 直到 $P_{i+1} = P_i$, 输出 (N, P_i).

3.6 连分式方法(CFRAC)

连分式方法(Continued FRACtion)是由 Morrison, Brillhart[127] 于 1975 年提出的, 他们运用此方法成功地分解了 Fermat 数 F_7. 它以及之后要介绍的二次筛(QS)以及数域筛(NFS)都基于如下一个简单的事实: 如果

$$x^2 \equiv y^2 \pmod{N}, \quad \text{且 } x \not\equiv y \pmod{N}.$$

则 $(N, x \pm y)$ 就是 N 的一个非平凡因子.

当然, 寻找这样的 x, y 不能只靠运气, CFRAC 方法要构造一组同余式

$$x_k^2 \equiv (-1)^{e_{0k}} p_1^{e_{1k}} \cdots p_m^{e_{mk}} \pmod{N}, \tag{3.1}$$

其中 p_i 都是因子基 FB 中较小的素数. 如果找到足够多这样的同余式(例如个数 $n > m + 1$), 那么利用二元域 $\mathbb{F}_2$ 上的 Gauss 消元法, 可以找到组合系数 $\varepsilon_k \in \mathbb{F}_2$ 使

得

$$\sum_{k=1}^{n} \varepsilon_k(e_{0k}, e_{1k}, \ldots, e_{mk}) \equiv (0, 0, \ldots, 0) \pmod 2.$$

我们记

$$(v_0, \ldots, v_m) = \frac{1}{2} \sum_{k=1}^{n} \varepsilon_k(e_{0k}, e_{1k}, \ldots, e_{mk}),$$

此时若令

$$x = \prod_{k=1}^{n} x_k^{\varepsilon_k}, \quad y = (-1)^{v_0} \prod_{i=1}^{m} p_i^{v_i}, \tag{3.2}$$

便有我们所需要的

$$x^2 \equiv y^2 \pmod N.$$

如何构造这么多同余式呢? 我们知道用连分式部分展式可以得到二次无理数 $\sqrt{KN}(K \in \mathbb{N})$的好的有理数逼近. 设 $\dfrac{P}{Q}$ 为其近似分数, 那么 $t = P^2 - KNQ^2$ 的绝对值就很小, 从而 t 很可能在因子基 FB 下分解, 同时 $P^2 \equiv t \pmod N$, 便能得到我们所期望的同余式 (3.1) .

算法3.6 (CFRAC 方法)**.**

1. 选择适当的 $K \in \mathbb{N}$(通常取为 1, 当连分式展式周期太小而无法产生足够的同余式时选择另一个 K), 令 $FB = \{p_1, p_2, \ldots p_m\}$, 使得 $\left(\dfrac{KN}{p_i}\right) \neq -1, i = 1, \ldots, m.$
2. 计算 $\sqrt{KN}$ 的连分式展式, 得到一系列近似分式 $\dfrac{P_k}{Q_k}$.
3. 计算 $t_k = P_k^2 - KNQ_k^2$, 尝试在 FB 下得到 t_k 的分解, 若分解成功则有
$$P_k^2 \equiv (-1)^{e_{0k}} \prod_{i=1}^{m} p_i^{e_{ik}}.$$
4. 当得到足够多的同余式时($n > m + 1$ 即可), 用 $\mathbb{F}_2$ 上的 Gauss 消元法得到 (3.2) 中的 x, y.
5. 若 $x \not\equiv \pm y \pmod N$, 输出 N 的非平凡因子 $(N, x \pm y)$.

注39. 由于 $(P_k, Q_k) = 1$, 因此若 $p_i \mid t_k = P_k^2 - KNQ_k^2$, 必有 $p_i \nmid Q_k$, 因此 KN 必定为模 p 的平方数, 从而第一步中可以只选择限定条件的素数 p_i.

注40. 连分式的计算可以只用简单的四则运算, Gauss 消元法可以用一些稀疏矩阵的专用算法来加速, 因此 CFRAC 最花时间的部分在 t_k 的分解上, 当分解 t_k 花去太久时间时可以直接放弃, 转而求下一个同余式.

注41. CFRAC 方法的时间复杂度为 $L_N[\frac{1}{2}, \sqrt{2}]$(见 [174]), 其中 L 记号如下定义:

$$L_N[\alpha, c] = O\left(\exp\left((c + o(1))(\ln N)^{\alpha}(\ln\ln N)^{1-\alpha}\right)\right).$$

注42. 寻找同余式 (3.1) 来进行分解的想法首先来自 Dixon[68], 他当时的做法是直接随机地选取 x, 然后在 FB 下分解 x^2, 算法复杂度为 $L_N[\frac{1}{2}, 2\sqrt{2}]$, 这也是第一个次指数阶的一般整数分解方法.

3.7 Lenstra 椭圆曲线方法(ECM)

因子分解说到底就是寻找 x, 使得 (x, N) 非平凡, 关键在于提高寻找的 x 的成功率. Pollard $p-1$ 方法通过计算 $a^{p-1}-1$ 来提高成功率, 实质上是在群 $\mathbb{Z}/p\mathbb{Z}^*$ 中考虑问题. Lenstra[118] 提出的椭圆曲线方法(Elliptic Curve Method)转而在有限域上随机的椭圆曲线群中考虑问题. 由于椭圆曲线可以有许多不同的选择, ECM方法要比 Pollard $p-1$ 高效许多, 到目前为止是第三快的因子分解方法, 仅次于数域筛和二次筛.

首先我们给出域上椭圆曲线的定义:

定义3.1 (域 F 上的椭圆曲线)**.** 设 F 是特征不为 2, 3 的域, $x^3 + ax + b \in F[x]$ 无平方因子, O 表示无穷远点, 则

$$E = \{(x, y) \in F^2 \mid y^2 = x^3 + ax + b\} \cup \{O\}$$

称为 F 上的一条椭圆曲线.

注43. $x^3 + ax + b$ 无平方因子等价于判别式 $-16(4a^3 + 27b^2) \neq 0$, 即椭圆曲线是非奇异的, 在几何上看没有“尖点”.

椭圆曲线既是代数曲线又是一个加法群:

定义3.2 (椭圆曲线上的加法运算)**.** 设 E 为一椭圆曲线, $P, Q \in E$, 过 P, Q 的直线交 E 于三点 $\{P, Q, S\}$. $-S$ 表示 S 关于 x 轴的对称点. 定义加法

$$P + Q = -S.$$

另有三种特殊约定:

1. 若 $P = Q$, 则直线视为 P 处的切线;
2. 若 $P = -Q$, 则定义 $P + Q$ 为无穷远点 O;
3. 若 $Q = O$, 则定义 $P + O = -(-P) = P$.

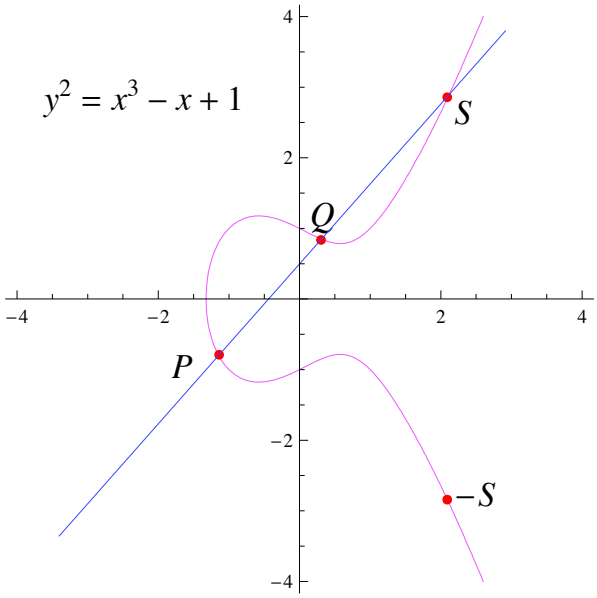

图 3.2: 椭圆曲线

由定义通过简单的计算我们可以得到:

命题3.1 (加法的显式表达). 设 $P = (x_1, y_1)$, $Q = (x_2, y_2)$, $P + Q = (x_3, y_3)$, 则

$$\begin{cases} x_3 = \lambda^2 - x_1 - x_2, \\ y_3 = \lambda(x_1 - x_3) - y_1. \end{cases} \tag{3.3}$$

其中

$$\lambda = \begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} & 若 P \neq Q, \\ \dfrac{3x_1^2 + a}{2y_1} & 若 P = Q. \end{cases}$$

注44. 可以知道以上加法定义确实使 E 成为了一个加法交换群(尽管结合性的验证需要一些繁琐的计算).

上面我们考虑了域上的椭圆曲线, 然而对于因子分解的任务来说, 我们需要考虑 $\mathbb{Z}/N\mathbb{Z}$ 上的椭圆曲线. 由于 $x_1 - x_2$ 等在 $\mathbb{Z}/N\mathbb{Z}$ 中未必可逆, 此时上面的加法运

算未必能定义好, 不过这无关紧要, 例如当 $x_1 - x_2$ 不可逆时, 我们已经可以通过计算 $(x_1 - x_2, N)$ 来得到 N 的非平凡因子, 从而直接完成分解的目标; 而当 $x_1 - x_2$ 可逆时, 一切可以正常按照上面的显式表达进行运算. 因此在这里我们不再花功夫用严格的语言来定义 $\mathbb{Z}/N\mathbb{Z}$ 上的椭圆曲线了.

下面我们将 Pollard $p-1$ 中类似的想法用在椭圆曲线中.

定义3.3 (整数的光滑性). 整数 n 称为是 B-光滑, 若其最大素因子 $P^+(n) \leq B$.

Pollard $p-1$ 方法的实质就是期望整数 $p-1$ 足够光滑而能在因子基 FB 下分解. 和 Pollard $p-1$ 方法中的想法类似, ECM 中首先从椭圆曲线 E 中随机取一点 P, 我们期望 P 的阶 n 是足够光滑的, 从而可以在 FB 下分解, 然后通过加法规则计算 nP(当然我们预先并不知道 n), 利用计算过程中出现的不可逆元, 求得 N 的一个因子.

算法3.7 (Lenstra ECM).

设 $(N, 6) = 1$, B 为给定的搜索极限, $p_1, \ldots, p_m$ 为 B 以下的所有素数.

1. 随机选取整数 a, E 为曲线 $y^2 = x^3 + ax + 1$.
2. 设 $P = (0, 1)$, $e_k = \lfloor \ln_{p_k} B \rfloor$, 根据式 (3.3) 递推地计算 $\left(\prod_{k=1}^{m} p_k^{e_k} \right) P$. 若计算过程中出现不可逆元 t, 则到第三步, 否则到第一步.
3. 计算 $d = (t, N)$. 如果 $d \neq N$ 则输出 d, 算法终止; 如果 $d = N$ 则到第一步.

下面我们谈一下搜索极限 B 的取法, 和 Pollard $p-1$ 方法中一样, 我们需要知道有限域 $\mathbb{Z}/p\mathbb{Z}$ 上群 E 的阶, 下面有限域上椭圆曲线最主要的定理归功于 Hasse, 告诉我们群 E 的阶在 p 左右(参见 [56]):

定理3.1 (Hasse). 设 E 为一有限域 $\mathbb{Z}/p\mathbb{Z}$ 上的椭圆曲线, 则 $|E| = p + 1 - a_p$, 其中 $|a_p| < 2\sqrt{p}$.

下面的定理则给出了关于光滑性的一个估计(参见 [56]):

定理3.2 (Canfield-Erdős-Pomerance). 记

$$\psi(x, y) = \#\{n < x \mid P^+(x) \leq y\},$$

其中 $P^+(x)$ 表示 x 的最大素因子, 再设

$$L(x) = \exp\left(\sqrt{\ln x \ln\ln x}\right),$$

则有估计

$$\psi(x, L(x)^a) = xL(x)^{-\frac{1}{2a}+o(1)}, \quad x \to +\infty.$$

其中 a 为正实数.

由上面两个定理我们可以得到选取 B 的一些信息, 设素数 $p \mid N$, 而 $B = L(p)^a$, 由定理 3.1 和 3.2 知道平均要试 $L(p)^{\frac{1}{2a}+o(1)}$ 条曲线可以得到一个阶为 B-光滑的椭圆曲线, 算法 3.7 计算总共需要 $L(p)^{a+\frac{1}{2a}+o(1)}$ 个群运算, 为使运算量最小, 因此可取 $a = \frac{1}{\sqrt{2}}$. 实践中 B 的选择依赖于时间的承受限度, 例如我们将搜索的素数因子限制在 10^{20} 以下, 那么可取 $B = 12000$(接近 $L(10^{20})^{1/\sqrt{2}}$).

由上面的讨论可以看出, ECM 的时间复杂度依赖于 N 的最小素因子 $P^-(N)$ 而非 N 本身(为 $L_{P^-(N)}[\frac{1}{2}, 1]$), 因此很适宜在试除法和 Pollard ρ 方法之后用 ECM 来找出较小的因子(10–20 个十进制位左右).

ECM 算法的效率很大程度取决于群运算的快慢, 最关键的是模 N 的求逆运算. 我们在本节最后给出 Montgomery[126] 的一个加速算法, 使得我们能够同时对多条椭圆曲线进行求逆运算.

算法3.8 (Montgomery).

设 $a_1, \ldots, a_k$ 为不能被 N 整除的整数, 本算法求出其逆 $b_1, \ldots, b_k$ 或给出 N 的一个非平凡因子.

1. 递推计算

$$\begin{aligned} c_1 &\equiv a_1 \pmod{N}, \\ c_2 &\equiv a_1 a_2 \pmod{N}, \\ &\vdots \\ c_k &\equiv a_1 \cdots a_k \pmod{N}. \end{aligned}$$

2. 施行一次扩展 Euclid 算法求出 (u, v, d) 满足 $d = (c_k, N)$, $uc_k + vN = d$.

   - 若 $d = 1$, 则 a_i 均有逆, 到第三步;
   - 若 $d > 1$, 依次计算 $(d, a_1), (d, a_2), \ldots$ 直到 $(d, a_i) > 1$, 输出 (d, a_i) 为 N 的一个非平凡因子.

3. 递推计算逆 b_i 并输出:

$$\begin{aligned} b_k &\equiv uc_{k-1} \pmod N, \\ b_{k-1} &\equiv (ua_k)c_{k-2} \pmod N, \\ &\vdots \\ b_1 &\equiv (ua_k \cdots a_2) \pmod N. \end{aligned}$$

3.8 二次筛法(QS)

二次筛法(Quadratic Seive)是由 Pomerance[142] 于 1982 年提出的, 直到 1993 年仍然是世界上渐进最快的通用大整数因子分解方法, 第一的位置后来被数域筛法(NFS)所取代, 不过对于 120 位以下的整数, 二次筛法还是要比数域筛法快一些.

3.8.1 单个多项式二次筛法(SPQS)

正如我们在 CFRAC 方法中提到的, QS 方法也要构造一组同余式 (3.1) , 但通过筛法避免了其中 y 在因子基 FB 下的分解, 而这种分解在不存在的情况下常常会大量消耗时间. 设 $Q(a) \in \mathbb{Z}[a]$, 若 $m \mid Q(a)$, 则不难验证对于任意 $k \in \mathbb{Z}$, 也有 $m \mid Q(a + k \cdot m)$. 于是我们找到了一系列数都有因子 m, 这样一个事实构成了 QS 方法的基础.

我们取 $y = Q(a)$, 为了使其与某个 x^2 模 N 同余, 且尽可能小以便在 FB 下分解, 考虑二次多项式 $Q(a) = L)(\lfloor\sqrt{N}\rfloor + a)^2 - N$, $x = \lfloor\sqrt{N}\rfloor + a$, 则 $x^2 \equiv Q(a) \pmod N$ 且 $Q(a)$ 为 $O(N^{1/2+\varepsilon})$ 的阶, 符合我们的要求. 接下来是筛法的过程: 对于 FB 中满足搜索极限 $B > p^e$ 的素数 p 及幂次 e, 首先求解方程 $z^2 \equiv N \pmod{p^e}$, 其解数(如果有解的话)

$$n(p,e) = \begin{cases} 1 & p = 2, e = 1, 2, \\ 4 & p = 2, e \geq 3, \\ 2 & p \geq 3. \end{cases}$$

并且解都可以快速地求得. 对方程的任一个解 z, 令 $a_0 = z - \lfloor\sqrt{N}\rfloor$, 则有 $p^k \mid Q(a_0)$. 给定一个搜索区间 I(通常很长), 则对任意与 a_0 差 p^e 的整数倍的 $a \in I$, $Q(a)$ 都有因子 p^e. 可用一张表储存区间 I 中每个整数对应的因子, 当对所

有 p 与 e 进行如上过程后, 通过检查表, 即可得到许多 I 中在 FB 下完全分解的整数了. 接下来的步骤则与 CFRAC 的后半部分完全相同.

注45. 实践过程中可以构造这样一张表, 若 $p^e \mid Q(a)$, 则对应 a 的表项增加 $\ln p$(这可预先计算), 最后检查表项若接近 $\ln Q(a)$, 即可知道 $Q(a)$ 在 FB 可完全分解, 此处的对数函数的计算可以不那么精确, 只要绝对误差在 1 以下即可.

注46. 在一些启发性假设下, 利用 Canfield-Erdős-Pomerance 的定理 3.2 , 可以知道 QS 的时间复杂度为 $L_N[\frac{1}{2}, 1]$, 与 ECM 差不多. 但由于筛法的运用, QS 的运算更简单一些, 实践中要快于 ECM, 除非是在 $P^-(N)$ 较小的情形.

3.8.2 多个多项式二次筛法(MPQS)

MPQS 是对上述只用一个二次多项式的 SPQS 方法的一个改进, 使用更多的二次多项式来减小 $Q(a)$ 的值, 从而减小 FB 的大小和搜索区间的长度. 考虑 $Q(a) = Aa^2 + 2Ba + C$ 形式的多项式($A > 0$), 配方得 $AQ(a) = (Aa+B)^2 - (B^2 - AC)$, 因此可选取系数使 $N \mid B^2 - AC$, 设 $x = Aa + b$, 则 $AQ(a) \equiv x^2 \pmod N$. 我们需要 $Q(a)$ 尽可能的小, 设搜索区间的长度 $|I| = 2M$, 自然地把 I 的中心设置在 $Q(a)$ 的极小点 $-\dfrac{B}{A}$ 处, 此时

$$|Q(-B/a)| = \frac{B^2 - AC}{A},$$

且 $Q(a)$ 在 I 上的最大值与最小值之差为

$$Q\left(-\frac{B}{A} + M\right) - Q\left(-\frac{B}{A}\right) = AM^2 - 2\frac{B^2 - AC}{A},$$

从而宜取 $B^2 - AC = N$, 且 A 接近 $\sqrt{2N}/M$. 因此选择系数的过程可以如下进行

1. 选择区间长度 M.
2. 选择接近于 $\sqrt{2N}/M$ 的素数 A.
3. 求解 $B^2 \equiv N \pmod A$(例如利用 Shanks 算法 12.2).
4. 令 $C = (B^2 - N)/A$.

接下来的步骤便是对这样选取的多个多项式进行筛法, 最终得到足够多的同余式进行 $\mathbb{F}_2$ 上的 Gauss 消元法.

注47. MPQS 的过程明显有着并行化的特性.

3.9 数域筛法(NFS)

数域筛法(Number Field Sieve)是由 [117] 在 1993 年提出的, 为目前渐进最快的通用因子分解方法, 其时间复杂度为 $L_N[\frac{1}{3}, c]$, 其中常数 c 依赖于不同的算法实现. 例如对于针对 $r^e - s$ 形式整数的特殊数域筛法(SNFS)有 $c = (\frac{32}{9})^{\frac{1}{3}}$, 而对于一般数域筛法(GNFS)有 $c = (\frac{64}{9})^{\frac{1}{3}}$. 对于 120 位以上的大数, NFS 是最强有力的分解算法. 例如互联网上的分布式大整数分解项目NFSNet[1]采用的便是此法. 由于数域筛法涉及代数数域理论, 需要较多的准备, 限于篇幅我们不再赘述, 感兴趣的读者可参看 [56], [8] 或其他相关文献.

[1] http://www.nfsnet.org/

4 基础数论算法

本章将讨论素数判定, 因子分解以外的许多基础数论算法, 包括快速求幂, 幂次检测, 整数的最大公因子(GCD), Legendre-Jacobi-Kronecker 符号, 中国剩余定理, 连分数展式及一些常用的数论函数的计算. 这些问题和算法不仅有其本身的趣味, 而且在计算机代数系统中也起着非常基础性的作用, 为许多其他更复杂的问题和算法所依赖.

阅读本章要求读者有基本的初等数论知识(例如 [10]), 本章的主要参考书为 [56] 和 [104].

4.1 快速求幂

4.1.1 二进方法

设 G 为一乘法群, $n \in \mathbb{Z}$(G 可只为乘法半群, 此时限定 $n \in \mathbb{N}$), 计算 $g \in G$ 的幂次 g^n 是在计算机代数系统中到处出现的运算(尤其是 $G = \mathbb{Z}/N\mathbb{Z}$ 的情形), 因此快速进行求幂起着非常基本的作用. 最平凡的方法需要 $n-1$ 个乘法, 不过我们有办法大幅地改进乘法的次数.

一个启发性的想法是二分策略. 例如我们要求 g^{11}, 而 11 的二进制表示为 $11 = 2^3 + 2 + 1$, 可以依次求出 g^1, g^2, g^4, g^8, 再计算 $g^3 = g^1 \cdot g^2$ 及 $g^{11} = g^3 \cdot g^8$, 总共只用了 5 次乘法便得到了结果. 这种方法被称为自右向左的二进方法(Right-left Binary). 另外还有一种自左向右的二进方法(Left-right Binary): 对于上面的例子依次求出 g^1, g^2, $g^4 = (g^2)^2$, $g^5 = g^4 \cdot g^1$, $g^{10} = (g^5)^2$, $g^{11} = g^{10} \cdot g^1$, 同样用 5 次乘

法得到结果.

算法4.1 (自右向左的二进方法).
设二进表示 $n=\sum_{k=0}^{m}\varepsilon_k 2^k$(假定 $n>0$, 否则考虑 $(g^{-1})^{|n|}$), $a=1$, 重复以下步骤, 第 k 步$(k=0,1,\ldots,m)$执行:

1. 计算 $g^{2^k}=(g^{2^{k-1}})^2$.
2. 若 $\varepsilon_k=1$, 则令 $a\leftarrow a\cdot g^{2^k}$.
3. 若 $k=m$, 输出 $g^n=a$, 算法终止.

注48. 实践中 n 的二进表达式不需要预先存储, 其计算(通过位操作)可以与 g^{2^k} 的计算同步进行.

注49. 不难验证 G 中进行求 n 次幂的乘法次数为 $m-1+\sum_{k=0}^{m}\varepsilon_k=O(\log n)$, 这要远小于平凡算法的 $n-1$.

算法4.2 (自左向右的二进方法).
设二进表示 $n=\sum_{k=0}^{m}\varepsilon_k 2^k$(假定 $n>0$, 否则考虑 $(g^{-1})^{|n|}$), $a=g$, 重复以下步骤, 第 k 步$(k=m-1,m-2,\ldots,0)$执行:

1. 若 $\varepsilon_k=0$, 则 $a\leftarrow a^2$.
2. 若 $\varepsilon_k=1$, 则 $a\leftarrow a^2$, $a\leftarrow a\cdot g$.
3. 若 $k=0$, 输出 $g^n=a$, 算法中止.

注50. 算法 4.1 与算法 4.2 求 n 次幂所用的乘法次数相同, 但在算法 4.2 第二步累乘时乘上的因子 g 要比算法 4.1 乘上的因子 g^{2^k} 小一些.

4.1.2 m 进方法, 窗口方法及加法链

二进方法的一个很自然的推广就是 m 进方法, 利用 n 的 m 进表示, 类似二进方法的平方而不断地进行 m 次幂. $m=2^k$ 的情形尤其简单, 因此在实践中也很常用. 不同于二进方法, m 进方法还需要额外储存 $g^2,g^3,\ldots,g^{m-1}$ 的值. 2^k 进方法的进一步改进是所谓窗口方法(Window Method), 取一个固定长度 $k+1$ 的“窗

口”, 用窗口自左向右扫描 n 的二进表达, 计算窗口内的乘幂(往往预先计算好 $g^3,\ldots,g^{2^k-1}$), 然后通过乘幂将窗口向右平移(遇零即为平方), 将下一个窗口内的数乘以已经得到的幂, 再重复平移过程. 例如考虑 $n=26235947428953663183191$. 用 8 进方法需要 102 次乘法, 而通过长度 4 的窗口方法可以减少到 93 次(参见 [77]), 包括预先的 2^4-1 次计算, 71 次平方和 14 次中间过程中的乘法. 下式表示了此窗口方法的过程. 其中除去最左边的一条, 剩下的 14 条下划线各代表一次中间过程的乘法:

$$\underline{1011}\,000\,\underline{111}\,00\,\underline{1}\,000000\,\underline{111}\,0\,\underline{1001}\,0\,\underline{1001}\,\underline{1101}\,0\,\underline{1}\,000000\,\underline{1011}\,\underline{11}\,00000\,\underline{1111}\,\underline{1001}\,\underline{1001}\,0\,\underline{101}\,0\,\underline{111}$$

二进方法的乘法次数较平凡算法有本质的减少, 一个很自然的问题是: 计算 n 次幂所需要的最少乘法次数是多少? 这等价于求 n 的加法链的最短长度 $l(n)$(参见 [104]). 构造最短长度的加法链是较为困难的问题, 在实践中只能构造“近似”最短的加法链. 一个构造方法是通过“幂树”, 其构造方法是: 首先在第一层置 1, 当置完 k 层后, 从左到右依次取第 k 层每个节点 n, 在其下附加节点 $n+1, n+a_1,\ldots,n+a_{k-1}=2n$, 其中 $1,a_1,a_2,\ldots,a_{k-1}$ 是树根到 n 的到 n 的通路上的点, 但已出现的节点不予添加. 下图显示了前 4 层幂树.

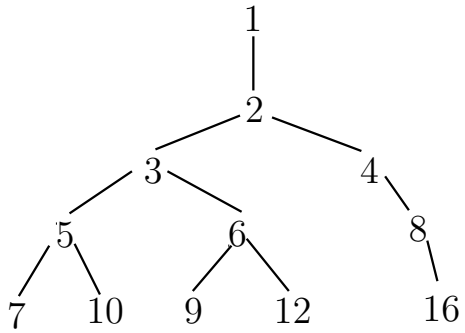

图 4.1: 前 4 层幂树

有了幂树之后, 当我们要计算某个 n 次幂时, 只需要在该树上找到 n, 依次计算从根节点到 n 的路径上的幂次即可.

幂树构造简单, 但对较大的 n 树的规模会很大(实际得到了 n 以下所有数的近似加法链). 有许多启发性的算法来构造直接 n 的近似最短加法链(参见 [77], [36]).

另外窗口方法和加法链也可以结合起来使用. 由于当窗口长度增加时, 需要预先计算的 g 的幂数量增多, 可以构造加法序列(含所需幂次的加法链)来减少预先的

计算量. 使用长度较大的窗口的确能够减少乘法的次数, 例如对前面的例子, 可以将 93 次乘法进一步减少到 89 次. 下式显示了长度为 8 的窗口方法过程.

$$\underset{5689}{1011000111001}\ 000000\ \underset{933}{1110100101}\ 00\ \underset{117}{1110101}\ 000000\ \underset{47}{101111}\ 00000\ \underset{499}{111110011}\ 00\ \underset{343}{101010111}$$

注51. 前面所述的乘幂算法对一般的乘法半群均适用(例如矩阵乘法, 多项式乘法). 考虑乘幂运算对象的特性, 还有更多可能改进的空间. 例如 G 为椭圆曲线的特殊情形(在这里是加法群), 由于元素的逆较为容易求得, 可以考虑形如 $n=\sum_{i\geq 0}\varepsilon_i 2^i\ (\varepsilon_i\in\{0,\pm 1\})$ 的冗余二进制表示, 来减少乘法次数. 对于 $G=\mathbb{Z}/N\mathbb{Z}$ 的特殊情形则有下面将要介绍的著名的 Montgomery 约化过程.

4.1.3 Montgomery 约化

乘幂运算最常见的是出现在 $G=\mathbb{Z}/N\mathbb{Z}$ 中. Montgomery[125] 提出一个避免试除的计算模乘法的方法, 对于 N 固定, 需要大量乘法的模幂运算尤其有效. 主要想法是将模 N 的运算转化为模机器字长(例如 2^{32})的运算, 而后者是硬件快速实现的.

定义4.1. 记 $\overline{x}=x \bmod N\in\{0,1,\ldots,N-1\}$, 即 x 模 N 的余数. 设 R 为一个进制的基(通常取为机器字长或其幂次), 满足 $(R,N)=1$, $R>N$. 则 $\exists 0\leq u<N$, $0\leq v<R$ 成立 Bezout 等式 $u\cdot R-v\cdot N=1$, 我们定义 $\widetilde{x}:=\overline{xu}$, 称为 x 的 Montgomery 表示.

注52. 由定义可知 $\overline{uR}=1$ 以及 $\overline{x}=\widetilde{xR}$. 并且当 x 跑遍模 N 的一个完全剩余系时, $\widetilde{x}$ 也跑遍模 N 的完全剩余系, 因此 $\widetilde{x}$ 这种表示不会损失信息.

下面的快速计算 $\widetilde{x}$ 的算法是 Montgomery 模幂乘法的核心.

算法4.3 (Montgomery 约化).

设 $0\leq x<RN$, 以下步骤计算 $\widetilde{x}$.

1. 计算 $m=(x \bmod R)\cdot v \bmod R$, $0\leq m<R$.
2. 计算 $t=(x+mN)/R$.
3. 若 $t\geq N$ 输出 $t-N$, 否则输出 t, 算法终止.

注53. 当 R 取为机器字长时, 第一, 二步中的模运算和除法运算都可直接通过位操作快速进行.

算法的有效性. 首先由 $x+mN \equiv x+xvN \equiv x(1+vN) \equiv xuR \equiv 0 \pmod{R}$ 知 t 为整数. 再由 $t=(x+mN)/R$, 知 $uRt=(xu+umN)$, 从而 $\overline{uRt}=\overline{xu}$, 即有 $\bar{t}=\tilde{x}$. 最后由 $0 \le x < RN$, $0 \le m < R$, 知 $0 \le t < 2N$, 可得算法的有效性. □

下面的命题表明, 在 Montgomery 表示下, $\widetilde{mn}$ 即表示 m 与 n 的乘积.

命题4.1. 设 $m, n \in \mathbb{Z}$, 则

$$\overline{\widetilde{m} \cdot \widetilde{n}} = \widetilde{\widetilde{m \cdot n}}.$$

证明. 由定义, $\overline{\widetilde{m} \cdot \widetilde{n}} = \overline{mu} \cdot \overline{nu} = \overline{mnu^2} = \widetilde{mnu} = \widetilde{\widetilde{m \cdot n}}$. □

为求 $\overline{x \cdot y}$, 首先将 $\bar{x}, \bar{y}$ 逆变换为 Montgomery 表示 $\widetilde{xR}, \widetilde{yR}$, 再利用命题 4.1 和算法 4.3 即可.

注54. 应用算法 4.3 时要注意保证输入满足在 0 和 RN 之间, 而 $xR \cdot yR$ 并不满足此条件. 可对两乘数先做一个模 N 处理, 这可利用 $\overline{xR} = \widetilde{x\overline{R^2}}$ 来完成, 而 $\overline{R^2}$ 作为一个固定的常数可预先计算好, 反复使用. 在求幂的过程中, 由于 R, N 固定, 只需要在第一步逆变换为 Montgomery 表示, 中间结果全部用 Montgomery 表示来运算(运用算法 4.1 或 4.2), 并对最终结果变换回正常表示即可, 由此加快了求幂的运算速度.

注55. 当 $(R, N) \neq 1$, 通常即 N 为偶数的情形, 模幂运算可通过将 N 分解为 $2^d \cdot N_0 (N_0$ 为奇数), 先对用算法 4.3 对 N_0 模幂, 再对 2^d 模幂(位操作本身就很快), 最后用中国剩余定理 4.16 重构出结果来.

注56. [87] 提出了比 Montgomery 方法更快的模幂算法, 执行效率约能提升 30%到 50%.

4.2 幂次检测

在许多算法中(例如 SQUFOF 算法 3.5), 需要判断一个整数是否是完全平方数(更一般地判断是否为素数幂), 如果是的话, 还要求出其平方根. 对于很大的整数, 这些算法都要更有针对性, 更高效一些才能满足需要.

4.2.1 整数开方

对于整数开方, 直接动用浮点数运算显然是低效而且不精确的, 但我们可以将经典的 Newton 迭代法做一个改进使之变得更经济.

算法4.4 (整数开方)**.**
输入整数 n, 以下算法计算 $\lfloor\sqrt{n}\rfloor$.

1. 令 $x_0 = n$.
2. 顺次计算 $x_{k+1} = \left\lfloor \dfrac{x_k + \lfloor n/x_k \rfloor}{2} \right\rfloor$.
3. 若 $x_{k+1} \geq x_k$, 输出 x_k, 算法终止, 否则跳到第二步.

注57. 算法中所有运算均为整数运算, 没有动用浮点数运算.

算法的有效性. 由

$$\left\lfloor \frac{x_{k-1} + \lfloor n/x_{k-1} \rfloor}{2} \right\rfloor > \frac{x_{k-1} + n/x_{k-1} - 1}{2} - \frac{1}{2} \geq \sqrt{n} - 1,$$

可知对于任意 k, 均有 $x_k \geq \lfloor\sqrt{n}\rfloor$. 若 $x_{k+1} \geq x_k$ 且 $x_k \neq \lfloor\sqrt{n}\rfloor$, 则必有 $x_k \geq \lfloor\sqrt{n}\rfloor + 1$. 于是

$$x_{k+1} - x_k = \left\lfloor \frac{\lfloor n/x_k \rfloor - x_k}{2} \right\rfloor < \left\lfloor \frac{n - x_k^2}{2x_k} \right\rfloor < 0,$$

矛盾! 从而必有 $x_k = \lfloor\sqrt{n}\rfloor$. □

4.2.2 平方检测

我们要判断给定的整数 n 是否是一个完全平方数, 直接用算法 4.4 计算 $\lfloor\sqrt{n}\rfloor^2$ 是一个办法, 不过我们在这样做之前可以首先排除许多非平方数的情形. 以下方法运用一个简单的事实: 如果 n 是一个平方数, 那么对任意整数 k, n 在 $\mathbb{Z}/k\mathbb{Z}$ 中都是一个平方数.

算法4.5 (平方检测)**.**
输入整数 n, 判断其是否为平方数, 若是, 求之.

1. 枚举生成数组 q_{64}, 满足

$$q_{64}[k]=\begin{cases}1 & \text{若}k\text{为}\mathbb{Z}/64\mathbb{Z}\text{中的平方数},\\ 0 & \text{若}k\text{为}\mathbb{Z}/64\mathbb{Z}\text{中的非平方数}.\end{cases}$$

   类似生成另外三个数组 q_{63}, q_{65}, q_{11}.

2. 若 $q_{64}[n \bmod 64]=0$, 输出非平方数, 算法终止; 否则计算 $r=n \bmod 45045(=63\times 65\times 11)$.

3. 若 $q_{63}[r \bmod 63]=0$ 或 $q_{65}[r \bmod 65]=0$ 或 $q_{11}[r \bmod 11]=0$, 输出非平方数, 算法终止.

4. 用算法 4.4 计算 $\lfloor\sqrt{n}\rfloor^2$, 若不等于 n 输出非平方数, 否则输出平方根, 算法终止.

注58. 若 n 非平方数, 则算法进行到最后一步的可能性非常小, 因为模 64, 63, 65, 11 的平方数个数分别为 12, 16, 21, 6, 从而这种情况发生的概率大约仅为为

$$\frac{12}{64}\cdot\frac{16}{63}\cdot\frac{21}{65}\cdot\frac{6}{11}=\frac{6}{715}.$$

我们从上式也可以看出选取四个数 64, 63, 65, 11 的缘由.

注59. 在实际算法中, 我们往往是通过实验来获取更佳的常数选择方案.

4.2.3 素数幂检测

我们有时会需要检测一个整数 n 是否为素数幂 p^k, 例如素数幂检测可以作为一般整数因子分解算法的子算法. 若 $n=p^k$, 则由 Fermat 小定理, 对于任意整数 a 都有 $p\mid a^p-a$, 从而 $p\mid(a^n-a,n)$.

例4.1. 取 $n=p^k$, 我们计算 (a^n-a,n):

1. $n=3^3$, 取 $a=2$, 则 $(a^n-a,n)=(2^{27}-2,27)=(134217726,27)=3$.

2. $n=5^2$, 取 $a=3$, 则 $(a^n-a,n)=(3^{25}-3,25)=(847288609440,25)=5$.

3. $n=2^6$, 取 $a=2$, 则 $(a^n-a,n)=(2^{64}-2,64)=(18446744073709551614,64)=2$.

但实际上通过试算几个例子可以发现大部分情况下都有 $(a^n-a,n)=p$. 这就启发我们得到如下的算法.

算法4.6 (素数幂检测).

输入整数 n, 判断其是否为素数幂, 若是, 输出素数 p.

1. 取 $a = 2$.
2. 用算法 4.3 计算 $b = a^n \bmod n$ 并计算 $p = (b - a, n)$.
3. 若 $p = 1$, 输出非素数幂, 算法终止; 否则用对 p 进行合性检测(例如 Rabin-Miller 检测, 算法 2.5), 若 p 为合数, 则令 $a \leftarrow a + 1$, 跳到第二步.
4. 对 p 进行素性检测(例如 Lehmer $N - 1$ 检测, 定理 2.1), 若 p 不一定为素数, 则令 $a \leftarrow a + 1$, 跳到第二步.
5. p 为素数, 依次计算 $p, p^2, \ldots, p^{2^k}$ 直到 $p^{2^k} > n$.
   - 若 $p^{2^{k-1}} \nmid n$, 输出非素数幂, 算法终止;
   - 否则令 $n \leftarrow \dfrac{n}{p^{2^{k-1}}}$, 重复第五步, 直到 $n = 1$.
6. 输出素数 p, 算法终止.

注60. 若 p 为素数, 第四步的跳转实际上很少发生.

4.3 最大公因子

相对于因子分解, 求两个整数 a, b 的最大公因子$\gcd(a, b)$(Greatest Common Divisor, GCD)要快上许多, 甚至往往超出我们的想象. 用普通的个人电脑在 2 秒以内计算两个十万位的整数的最大公因子也不是一件难事.

据 [91] 的数据, 一个典型的代数运算(例如求多元多项式组的 Gröbner 基)通常要花上一半以上时间用来计算某两个大整数的最大公因子, 或许正因为最大公因子在计算机代数系统中的重要性, 对它的研究才能如此深入.

为了避免混淆, 在本节中始终用 $\gcd(a, b)$ 代表最大公因子, 而出现在算法的描述中的 (a, b) 则仅仅代表二元有序数对.

4.3.1 Euclid 算法

最平凡的求整数 a, b 的 GCD 的办法是直接将两数分解, 但这实在是太慢了, 只在 a, b 都小于 100 或者已知其中之一为素数的时候才可能有些用处. 早在古希

腊时代人们就已经知道如何进行更高效的计算了, 经典的 Euclid 算法虽最古老却也是最本质最有效的方法之一.

算法4.7 (Euclid).

输入正整数 a, b, 计算 $\gcd(a,b)$.

1. 若 $b=0$, 则输出 a, 算法终止.
2. 令 $(a,b)\leftarrow(b,a \bmod b)$, 跳到第一步.

Knuth[104] 通过一系列分析给出了 Euclid 算法终止前除法步数的精确估计.

定理4.1 (Euclid 算法的步数). *设 a, b 随机分布于 $[1,N]$, 则*

1. *Euclid 算法的平均步数为* $\dfrac{12\ln 2}{\pi^2}\ln N+O(1)\approx 0.843\ln N+O(1)$.
2. *Euclid 算法在最坏情况下的步数至多为* $\lfloor \log_\phi(3-\phi)N\rfloor \approx 2.078\ln N+0.6723$, *其中* $\phi=\frac{1+\sqrt{5}}{2}$.

4.3.2 Lehmer 加速算法

由定理 4.1 可见 GCD 可以在多项式时间内求出, 但 Euclid 算法中每一步的大整数试除还是相当耗费时间, Lehmer 针对这一点给出了加速的方法, 其基本想法是试除的商 $\left\lfloor \dfrac{a}{b}\right\rfloor$ 很大程度上只与 a, b 的前若干位有关, 从而可以用单精度的运算代替高精度的大整数试除.

例4.2. 我们想要计算 $\gcd(a,b)=\gcd(27182818,10000000)$, 假设计算机的字长为 4 个十进制数字, 令 $\hat{a}=2718$, $\hat{b}=1000$, 对 $\hat{a}+1$, $\hat{b}$ 进行 Euclid 算法的试商结果很可能与 a, b 进行的结果是一致的, 如下表.

a	b	q	$\hat{a}+1$	$\hat{b}$	$\hat{q}$
27182818	10000000	2	2719	1000	2
10000000	7182818	1	1000	719	1
7182818	2817182	2	719	281	2
2817182	1548454	1	281	157	1
1548454	1268728	1	157	124	1
1268728	279726	1	124	33	3

我们看到直到第 6 步试除, q 与 $\hat{q}$ 才不相等, 因此可以利用 a 与 b 的前若干位来快速求得试除的商. 一个存在问题是如何判断试商的正确性? 解决方法是可以同时求另一组 $\gcd(\hat{a}, \hat{b}+1) = \gcd(2718, 1001)$ 的试商.

$\hat{a}+1$	$\hat{b}$	$\hat{q}$	$\hat{a}$	$\hat{b}+1$	$\hat{q}$
2719	1000	2	2178	1001	2
1000	719	1	1001	716	1
719	281	2	716	285	2
281	157	1	285	146	1
157	124	1	146	139	1
124	33	3	139	7	19

显然若两组求得的商相同, 即可判定试商是准确无误的. 上表也验证了这一点.

算法4.8 (Lehmer 加速算法)**.**

输入正整数 a, b, 且 $a > b$, 计算 $\gcd(a, b)$. 设 M 为单精度整数的上限(例如 2^{32}), 以下 $\hat{a}$, $\hat{b}$, A, B, C, D 均为单精度整数(即 $< M$).

1. 若 $b < M$, 直接用 Euclid 算法计算输出 $\gcd(a, b)$, 算法终止; 若 $b \geq M$, 记 $\hat{a}$, $\hat{b}$ 为 M 进制下 a, b 的最高位, $\begin{pmatrix} A & C \\ B & D \end{pmatrix} \leftarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.
2. 若 $\hat{b}+C \neq 0$, $\hat{b}+D \neq 0$, 则令 $q \leftarrow \left\lfloor \dfrac{\hat{a}+A}{\hat{b}+C} \right\rfloor$, 否则跳到跳到第四步. 若还有 $q = \left\lfloor \dfrac{\hat{a}+B}{\hat{b}+D} \right\rfloor$, 则跳到第三步, 否则跳到第四步.
3. 令 $\begin{pmatrix} A & C \\ B & D \end{pmatrix} \leftarrow \begin{pmatrix} A & C \\ B & D \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix}$, $(\hat{a}, \hat{b}) \leftarrow (\hat{a}, \hat{b}) \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix}$, 跳到第二步.
4. 若 $B \neq 0$, 令 $(a, b) \leftarrow (a, b) \begin{pmatrix} A & C \\ B & D \end{pmatrix}$, 跳到第一步; 否则用高精度运算直接求出 $(a, b) \leftarrow (b, a \bmod b)$, 跳到第一步.

注61. Lehmer 的加速算法虽然步数和普通的 Euclid 算法是一样的, 但除了第四步后一种情形外, 都只需要单精度运算或单精度与高精度数的乘法, 这起到了“加速”的作用.

注62. 注意算法的陈述中忽略了一种特殊情形, 即当 $\hat{a} = M-1, A = 1$ 或 $\hat{b} = M-1, D = 1$ 时, 计算可能超出单精度的范围, 可以将 M 取为略小与单精度整数上限即可.

4.3.3 二进方法(Binary GCD)

另一种二进方法(Binary GCD)在实践中也很有用, 其主要想法是用减法和移位来代替代价较大的试除. 尽管需要的运算步数增多了, 但由于减法和位移运算的简单性节省出的是时间还是很可观的.

下面的定理精确地叙述了只用减法的合理性(参考 [104]).

定理4.2. *设 b 固定, a 随机分布. 对 a, b 施行 Euclid 算法, 其中间步骤试商为 q 的概率记为 $P_b(q)$, 则有*

$$\lim_{b\to\infty} P_b(q) = \log\frac{(q+1)^2}{(q+1)^2-1}.$$

注63. 由定理 4.2 , 粗略地估计试商值为 1 的概率为 $\log\frac{4}{3} \approx 0.415$, 试商值为 2 的概率为 $\log\frac{9}{8} \approx 0.170$. 可见大部分情况下除数和被除数都是比较接近的, 可以用减法来替代试除.

算法4.9 (Binary GCD).

输入正整数 a, b, 且 $a > b$, 计算 $\gcd(a, b)$.

1. 用高精度直接计算 $(a, b) \leftarrow (b, a \bmod b)$.
2. 若 $b = 0$, 输出 a, 算法终止; 否则设 $a = 2^m a_0$, $b = 2^n b_0$, a_0, b_0 为奇数. 设 $k = \min\{m, n\}$, 通过位操作计算 $(a, b) \leftarrow (a \cdot 2^{-m}, b \cdot 2^{-n})$.
3. 令 $t \leftarrow a - b$, 若 $t = 0$, 输出 $a \cdot 2^k$, 算法终止.
4. 设 $t = 2^s t_0$, t_0 为奇数, 若 $t_0 > 0$, 令 $a \leftarrow t_0$, 否则令 $b \leftarrow -t_0$, 跳到第三步.

注64. 由于初始的 a 和 b 可能相差很大, 不适宜反复做减法, 因此还是需要第一步的高精度试除.

注65. Gosper 提出了一个将 Binary GCD 想法与 Lehmer 加速算法结合起来的方法, 参见 [104].

4.3.4 扩展 Euclid 算法

在很多问题中(例如求模 N 的逆)不仅需要知道 $\gcd(a,b)$, 还要求出 Bezout 等式中的系数 u, v, 使得成立

$$ua+vb=d(:=\gcd(a,b)).$$

这样的算法称为扩展 Euclid 算法. 在原始的 Euclid 算法过程中保留所有的商和余数即可倒推出 u 和 v 来, 不过这样做需要较大的空间开销. 可以在计算过程中保留类似于 Lehmer 加速算法中的矩阵 $\begin{pmatrix}A & C\\ B & D\end{pmatrix}$, 矩阵的列实际上就是新操作数作为 a, b 线性组合的系数.

Lehmer 加速算法本身就就具有了“扩展”的特性, 但 Binary GCD 的“扩展”能力表面上看来就没有那么直接了. 不过仔细追踪算法的行进过程, 还是能够求出系数 u, v 来的.

算法4.10 (扩展 Binary GCD)**.**

输入正整数 a, b, 且 $a>b$, 计算三元组 (u,v,d).

1. 用高精度直接计算 $(a,b)\leftarrow(b,a \bmod b)$.
2. 若 $b=0$, 输出 $(0,1,b)$, 算法终止; 否则设 $a=2^m a_0$, $b=2^n b_0$, a_0, b_0 为奇数. 设 $k=\min\{m,n\}$, 通过位操作计算 $(a,b)\leftarrow(a\cdot 2^{-k}, b\cdot 2^{-k})$.
3. 令 $(u_1,v_1,d_1)\leftarrow(1,0,a)$, $(u_2,v_2,d_2)\leftarrow(b,1-a,b)$. 若 a 为奇数, 令 $(u_0,v_0,d_0)\leftarrow(0,-1,-b)$, 跳到第五步; 否则令 $(u_0,v_0,d_0)\leftarrow(1,0,a)$.
4. 若 u_0,v_0 均为偶数, 令 $(u_0,v_0,d_0)\leftarrow(u_0,v_0,d_0)/2$; 否则令 $(u_0,v_0,d_0)\leftarrow(u_0+b,v_0-a,d_0)/2$.
5. 若 d_0 为偶数, 跳到第四步.
6. 若 $d_0>0$, 令 $(u_1,v_1,d_1)\leftarrow(u_0,v_0,d_0)$; 否则令 $(u_2,v_2,d_2)\leftarrow(b-u_0,-a-v_0,-d_0)$.
7. 令 $(u_0,v_0,d_0)\leftarrow(u_1-u_2,v_1-v_2,d_1-d_2)$, 若 $u_0\le 0$, 令 $(u_0,v_0)\leftarrow(u_0+b,v_0-a)$. 若 $d_0=0$, 输出 $(u_1,v_1,d_1\cdot 2^k)$, 算法终止; 否则跳到第四步.

算法的有效性. 可以仔细验证每一步后都满足 $0\le u_1,u_2,u_0\le b$, $-a\le v_1,v_2,v_0\le 0$, $0<d_1\le a$, $0<d_2\le b$, 且满足 $u_i a+v_i b=d_i$, 对 $i=1,2,3$. 算法的终止性是明

显的. □

注66. 与原始的 Binary GCD 有区别的是第四步, 这是为了保证系数也能除以 2 而做的微小改动.

4.3.5 dmod 与 bmod

首先我们引进某种程度上相当于 mod 的运算 dmod(digit mod)和 bmod(bit mod).

定义4.2 (dmod 和 bmod)**.** 设 β 为一个进制的基, 正整数 a, b, 满足 $a > b$, $\gcd(a,\beta) = 1$, $\gcd(b,\beta) = 1$, $l_\beta(a)$ 表示 a 在 β-进制表示下的位数, 记 $\delta = l_\beta(a) - l_\beta(b) + 1$. 定义

$$a \operatorname{dmod}_\beta b := \frac{|a - (ab^{-1} \bmod \beta^\delta)b|}{\beta^\delta}.$$

当 $\beta = 2$ 时(最常用的情形), 简记为

$$a \operatorname{bmod} b := a \operatorname{dmod}_2 b.$$

注67. 由于 $a - (ab^{-1} \bmod \beta^\delta)b \equiv a - a \equiv 0 \pmod{\beta^\delta}$, 故 dmod 和 bmod 是可以定义好的.

例4.3. 设 $a = 155 = (10011011)_2$, $b = 15 = (1111)_2$, 则 $l_2(a) = 8$, $l_2(b) = 4$, $2^\delta = 2^{l_2(a)-l_2(b)+1} = 32$, $15^{-1} \equiv 15 \pmod{32}$. 从而

$$\begin{aligned} 155 \operatorname{bmod} 15 &= \frac{|155 - (155 \cdot 15 \bmod 32) \cdot 15|}{32} \\ &= \frac{|155 - 21 \cdot 15|}{32} = 5. \end{aligned}$$

恰巧有 $\gcd(155, 15) = 5$.

可以直接验证 $\gcd(a \operatorname{dmod}_\beta b, b) = \gcd(a,b)$, 而且 $a \operatorname{dmod}_\beta b < b$, 因此新定义的运算的确在某种程度上相当于 mod. 而且我们可以避免试除, 只用减法和位操作来计算出 $a \operatorname{bmod} b$. 最关键的是如何快速计算 $m = ab^{-1} \bmod 2^\delta$, 接下来的算法只用减法和位操作解决了这个问题, 想法类似于待定系数法求解 $bm \equiv a \pmod{2^\delta}$, 从低到高将 m 的二进表达依次求出.

算法4.11 (计算 $ab^{-1} \bmod 2^\delta$)**.**

输入正奇数 a, b, 计算 $ab^{-1} \bmod 2^\delta$.

1. 令 $d \leftarrow a$, $m \leftarrow 0$.
2. 重复以下步骤($i = 0, 1, \ldots \delta - 1$): 若 $2^{i+1} \nmid d$, 则令 $d \leftarrow d - 2^i \cdot b$, $m \leftarrow m + 2^i$.
3. 输出 m, 算法终止.

算法的有效性. 每次第二步的操作后均有 $d = a - mb \equiv 0 \pmod{2^{i+1}}$, 可知算法的有效性. □

注68. Weber[180] 仅仅采用 bmod 代替 Euclid 算法中的 mod 操作, 修改得到的算法比 Binary GCD 还要快一些.

注69. 设 $M = 2^w$ 为单精度整数的上界, 当 $a \gg b$, $l_2(a) - l_2(b) + 1 \gg w$ 时, 可以将 bmod 操作限制在 M 下以提高速度, 即取 $\delta = w$, 降低 $l_2(a)$ 若干次直到 $l_2(a) - l_2(b) + 1 \leq w$.

4.3.6 Jebelean-Weber-Sorenson 加速算法

Jebelean-Weber-Sorenson 加速算法的基本想法来自 Sorenson[164]. 设 a, b 与 n 互素, 如果存在 u, v 满足 $|u|, |v| < \sqrt{n}$ 使得 $ua \equiv vb \pmod{n}$, 为求 $\gcd(a, b)$, 考虑新的问题 $\gcd(\frac{ua-vb}{n}, b)$ 可以将 a 缩小大约 $\sqrt{n}$ 倍(这类似于 Binary GCD 中除以 2 的过程, 被称为 n-约化). 这样的 u, v 很可能是存在的, 例如考虑满足 $|u_0|, |v_0| < \sqrt{n}/2$ 二元组 (u_0, v_0), 这样的二元组共有 n 个, 计算 $u_0 a - v_0 b$ 其中必有两组模 n 同余, 将其相减即得满足条件的 (u, v). 下面的算法严格地构造出了 (u, v).

算法4.12.

输入正整数 a, b, n, 满足 a, b 与 n 互素, 计算 (u, v) 满足 $0 < u, |v| < \sqrt{n}$ 且 $ua \equiv vb \pmod{n}$.

1. 计算 $c \leftarrow ab^{-1} \pmod{n}$(当 n 为 2 的幂时可使用算法 4.11).
2. 令 $(u_1, v_1) \leftarrow (n, 0)$, $(u_2, v_2) \leftarrow (c, 1)$.
3. 若 $u_2 \geq \sqrt{n}$, 则令 $(u_1, v_1) \leftarrow (u_1, v_1) - \left\lfloor \frac{u_1}{u_2} \right\rfloor (u_2, v_2)$, 交换 (u_1, v_1) 与 (u_2, v_2), 跳到第三步.
4. 输出 (u_2, v_2), 算法终止.

算法的有效性. 算法本质上就是一个 Euclid 算法. 由于正数 u_1 的严格递降, 算法必在有限步后终止. 终止时有 $u_2 < \sqrt{n} \leq u_1$, 而算法过程中始终保持 $u_1|v_2| + u_2|v_1| = n$, 可知 $u_1|v_2| < n$, 从而 $|v_2| < \sqrt{n}$. □

注70. 算法 4.12 的第三步还可以用减法和位操作来避免试除, 当 u, v 相差不大时更加高效.

现在我们可以高效的计算 $\gcd(\frac{ua-vb}{n}, b)$ 了, 不过还有一个问题, 一般来说 $\gcd(a,b) \mid \gcd(\frac{ua-vb}{n}, b)$, 但两者未必相等. 如果直接计算前者, 可能结果会包含"假因子". 通常情况下可以在最后一步处理假因子, 因为根据 Dirichlet 的优美定理(参考 [104]), 通常求出的最大公因子都会很小(≤ 3 的比例为 82.7%), 最后一步的处理只花很少时间.

定理4.3 (Dirichlet, 最大公因子的分布)**.** *设 $q_n(d)$ 表示在 $1 \leq a, b \leq n$ 范围内使 $\gcd(a,b) \leq d$ 的有序数对 (a,b) 的个数, 则有*

$$\lim_{n\to\infty} \frac{q_n(d)}{n^2} = \frac{6}{\pi^2} \sum_{k=1}^{d} \frac{1}{k^2}.$$

算法4.13 (Jebelean-Weber-Sorenson 加速算法)**.**

设正整数 a_0, b_0 满足 $a_0 > b_0$ 且 β 互素, 依据 b_0 的长度选取 $s(b)$ 与 $t(b)$(例如当 $\beta = 2$ 可固定为 10 与 32).

1. 令 $a \leftarrow a_0$, $b \leftarrow b_0$.
2. 若 $b = 0$, 跳到第五步.
3. 若 $l_\beta(a) - l_\beta(b) > s(b)$, 则令 $a \leftarrow a \operatorname{dmod}_\beta b$, 跳到第三步; 否则根据算法 4.12 求得 (u,v) 使 $ua \equiv vb \pmod{\beta^{2t(b)}}$, 令 $a \leftarrow \dfrac{|ua - vb|}{\beta^{2t(b)}}$.
4. 将 a 中的 β 因子去除(当 $\beta = 2$ 时移位即可), 并交换 a 与 b, 跳到第二步.
5. 计算 $x = \gcd(a, b_0 \operatorname{dmod}_\beta a)$, 计算输出 $\gcd(x, a_0 \operatorname{dmod}_\beta x)$, 算法终止.

注71. 第五步相当于用 dmod 运算快速求得了 $\gcd(\gcd(a, b_0), a_0)$.

注72. 根据 Weber[180] 的实验结果, 在 8 个单精度整数长度时(大约 77 个十进制位), Jebelean-Weber-Sorenson 加速算法开始超过 Binary GCD.

尽管一般来说“假因子”都很小, 但也有例外的时候, Sedjelmaci[157] 以相邻的 Fibonacci 数 (F_N, F_{N+1}) 为例, 当 $N = 300$ 时, 假因子为 5, $N = 2000$ 时, 假因子为 122542875, 而当 $N = 3000$ 时, 假因子已高达 1.02×10^{15}, 而实际上相邻的 Fibonacci 数总是互素的.

不过 Sedjelmaci 给出了一个改进来避免假因子的影响, 这依赖于以下他证明的命题.

命题4.2 (Sedjelmaci)**.** 设算法 *4.12* 终止时得到了 (u_1, v_1), (u_2, v_2), 若有 $\frac{a}{b} < \sqrt{n}$, 则满足

$$\gcd(a, b) = \gcd\left(\frac{|u_1 a - v_1 b|}{n}, \frac{|u_2 a - v_2 b|}{n}\right).$$

因此可以的到改进版的 Jebelean-Weber-Sorenson 加速算法:

算法4.14 (Sedjelmaci 改进算法)**.**

设正整数 a_0, b_0 满足 $a_0 > b_0$ 且 β 互素, 依据 b_0 的长度选取与 $t(b)$(例如当 $\beta = 2$ 可固定为 32).

1. 令 $a \leftarrow a_0$, $b \leftarrow b_0$.
2. 若 $ab = 0$, 跳到第五步; 否则令 $(a, b) \leftarrow (\max\{a, b\}, \min\{a, b\})$.
3. 若 $\frac{a}{b} \geq \beta^{t(b)}$, 则令 $(a, b) \leftarrow (b, a \bmod_{\beta} b)$; 否则根据算法 4.12 求得 (u_1, v_1), (u_2, v_2), 使 $u_2 a \equiv v_2 b \pmod{\beta^{2t(b)}}$, 令 $(a, b) \leftarrow \left(\frac{|u_1 a - v_1 b|}{n}, \frac{|u_2 a - v_2 b|}{n}\right)$.
4. 将 a, b 中的 β 因子去除(当 $\beta = 2$ 时移位即可), 跳到第二步.
5. 输出 a, b 中不为零的数, 算法终止.

注73. 在算法的第三步可能牺牲了一些时间, 但保证了最终得到的最大公因子是不掺“假”的.

4.4 Legendre-Jacobi-Kronecker 符号

Legendre 符号不仅本身很重要, 而且在素性检测, 因子分解等领域中也经常被用到.

定义4.3 (Legendre 符号)**.** 设 p 为奇素数, $a \in \mathbb{Z}$, Legendre 符号定义为

$$\left(\frac{a}{p}\right) := \begin{cases} 1 & a\text{为模}p\text{的二次剩余}, \\ -1 & a\text{为模}p\text{的二次非剩余}, \\ 0 & p \mid a. \end{cases}$$

下面的性质是我们在初等数论教程中熟知的.

命题4.3 (Legendre 符号的基本性质)**.** *Legendre 符号满足:*

1. *(积性)*
$$\left(\frac{a}{p}\right)\left(\frac{b}{p}\right) = \left(\frac{ab}{p}\right).$$
2. *若* $a \equiv b \pmod{p}$, *则*
$$\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right).$$
3. *(Euler 准则)*
$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod{p}.$$

利用命题 4.3 中的第三条便可以通过模幂运算用来计算 Legendre 符号, 时间复杂度为 $O(\log^3 p)$. 对此方法本质上的改进依赖于著名的二次互反律.

定理4.4 (二次互反律)**.** 1. *若* p, q *为不相等的素数, 则*

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{\frac{(p-1)(q-1)}{4}}.$$

2. *(二次互反律的补充)*

$$\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}, \quad \left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}.$$

利用命题 4.3 中的第一, 二条及二次互反律可以得到一个直接的计算 $\left(\frac{a}{p}\right)$ 的方法(这也是我们在初等数论课程中掌握的). 不过此方法依赖于 a 以及某些中间结果的因子分解, 而分解往往不是一件易事. 但幸运的是我们还有更好的方法, 可以将时间复杂度降到 $O(\log^2 p)$, 这依赖于 Legendre 符号的推广 — Kronecker-Jacobi 符号 .

定义4.4 (Kronecker-Jacobi 符号)**.** 设 $a, b \in \mathbb{Z}$, 定义 Kronecker-Jacobi 符号:

1. 若 $b=-1$,
$$\left(\frac{a}{-1}\right):=\begin{cases}1 & a\geq 0,\\ -1 & a<0.\end{cases}$$

2. 若 $b=2$,
$$\left(\frac{a}{2}\right):=\begin{cases}0 & 2\mid a,\\ (-1)^{\frac{a^2-1}{8}} & 2\nmid a.\end{cases}$$

3. 若 $b=1$,
$$\left(\frac{a}{1}\right):=1.$$

4. 若 b 为奇素数, $\left(\frac{a}{b}\right)$ 定义为 Legendre 符号.

5. 设 $b=\varepsilon\cdot\prod\limits_{i=1}^{m}p_i^{e_i}$, 其中 $\varepsilon=\pm1$, p_i 为素数, 根据前四条定义
$$\left(\frac{a}{b}\right):=\left(\frac{a}{\varepsilon}\right)\prod_{i=1}^{m}\left(\frac{a}{p_i}\right)^{e_i}.$$

6. 约定当 $b=0$ 时,
$$\left(\frac{a}{0}\right):=\begin{cases}1 & a=\pm1,\\ 0 & a\neq\pm1.\end{cases}$$

注74. 设 b 为奇数, 和 Legendre 符号一致, 若 $\left(\frac{a}{b}\right)=-1$ 则 a 必为模 b 的二次非剩余. 但需要注意的是, $\left(\frac{a}{b}\right)=1$ 并不意味着 a 为模 b 的二次剩余.

Kronecker-Jacobi 符号将 Legendre 符号推广到了所有整数的情形, 并且成立以下良好的性质.

定理4.5 (Kronecker-Jacobi 符号的基本性质)**.** *如下几条成立:*

1. $\left(\frac{a}{b}\right)=0$ *当且仅当* $(a,b)\neq1$.

2. *对任意* $a,b,c\in\mathbb{Z}$, *有*
$$\left(\frac{ab}{c}\right)=\left(\frac{a}{c}\right)\left(\frac{b}{c}\right).$$
若还有 $b\neq0,c\neq0$, *则*
$$\left(\frac{a}{bc}\right)=\left(\frac{a}{b}\right)\left(\frac{a}{c}\right).$$

3. 设 $b>0$, x 为任意整数. 若 $b\not\equiv 2 \pmod 4$, 则

$$\left(\frac{x+b}{b}\right)=\left(\frac{x}{b}\right),$$

若 $b\equiv 2 \pmod 4$, 则

$$\left(\frac{x+4b}{b}\right)=\left(\frac{x}{b}\right).$$

4. 设 $a\neq 0$, x 为任意整数. 若 $a\equiv 0,1 \pmod 4$, 则

$$\left(\frac{a}{x+|a|}\right)=\left(\frac{a}{x}\right),$$

若 $a\equiv 2,3 \pmod 4$, 则

$$\left(\frac{a}{x+4|a|}\right)=\left(\frac{a}{x}\right).$$

5. 设 a, b 为正奇数, 则有二次互反律:

$$\left(\frac{a}{b}\right)\left(\frac{b}{a}\right)=(-1)^{\frac{(a-1)(b-1)}{4}},$$

且

$$\left(\frac{-1}{b}\right)=(-1)^{\frac{n-1}{2}},\quad \left(\frac{2}{b}\right)=(-1)^{\frac{n^2-1}{8}}.$$

有了如上更一般的二次互反律, 我们可以得到计算 Legendre-Jacobi-Kronecker 符号的算法了.

算法4.15 (Legendre-Jacobi-Kronecker 符号).

输入 $a,b\in\mathbb{Z}$, 计算 $\left(\frac{a}{b}\right)$.

1. 若 $b=0$, 输出 1, 算法终止; 若 a, b 均为偶数, 输出 0, 算法终止.
2. 设 $b=\varepsilon\cdot 2^d\cdot b_0$, 使 b_0 为正奇数. 由定义直接计算 $m=\left(\frac{a}{\varepsilon}\right)\left(\frac{a}{2}\right)^d$.
3. 若 $a=0$, 由定义直接计算 $m\leftarrow\left(\frac{0}{b_0}\right)\cdot m$, 输出 m, 算法终止; 若 $a\neq 0$, 设 $a=\epsilon\cdot 2^e a_0$, a_0 为正奇数, 计算 $m\leftarrow\left(\frac{\epsilon}{b_0}\right)\left(\frac{2}{b_0}\right)^e\cdot m$.
4. 运用二次互反律递归地求

$$\left(\frac{a_0}{b_0}\right)=\left(\frac{b_0 \bmod a_0}{a_0}\right)(-1)^{\frac{(a_0-1)(b_0-1)}{4}}.$$

计算 $m\leftarrow\left(\frac{a_0}{b_0}\right)\cdot m$, 输出 m, 算法终止.

注75. 在算法的第四步, 我们保证了 a_0 为奇数, 从而可以使用定理 4.5 的第三条.

注76. $(-1)^{\frac{a-1}{2}}$, $(-1)^{\frac{a^2-1}{8}}$ 和 $(-1)^{\frac{(a-1)(b-1)}{4}}$ 均可通过位操作和查表获得, 例如(& 表示位与操作):

$$(-1)^{\frac{(a-1)(b-1)}{4}} = \begin{cases} -1 & \text{若 a\&b\&2,} \\ 1 & \text{其它情况.} \end{cases}$$

注77. 可以看出算法 4.15 与 Euclid 算法的复杂度相同, 为 $O(\log^2 n)$, 其中 n 为 a, b 的一个上界, 比直接用 Legendre 符号计算要高效一些.

注78. 算法 4.15 中第四步也可以用类似于 Binary GCD 的方法, 不用 Euclid 算法计算 $\left(\frac{b_0 \bmod a_0}{a_0}\right)$, 而转而计算 $\left(\frac{b_0 - a_0}{a_0}\right)$, 分离 $b_0 - a_0$ 中 2 的幂, 再用二次互反律. 当 a_0, b_0 较大时, 也可采用 bmod 操作.

4.5 中国剩余定理

中国剩余定理是为数不多以“中国”命名的重要定理 — 关于交换幺环结构的经典结果. 为了完整性, 我们以常用的 $\mathbb{Z}$ 上的语言叙述如下.

算法4.16 (中国剩余定理)**.**

设整数 $m_i (i = 1, \ldots, n)$两两互素, 则对任意的 $x_i (i = 1, \ldots, n)$, 存在唯一的整数 x(在模 $M = \prod_{i=1}^{n} m_i$ 的意义下), 满足 $x \equiv x_i \pmod{m_i} (i = 1, \ldots, n)$.

证明. 记 $M_i = M/m_i$, 则对于任意 i, 存在 a_i, 满足 $a_i M_i \equiv 1 \pmod{m_i}$, 从而 $x = \sum_{i=1}^{n} a_i M_i x_i$ 即满足条件. 唯一性是显然的. □

这个构造性的证明可以得到一个直接的中国剩余定理算法, 不过如果 M_i 很大的话, 直接求逆或许不那么划算, 我们可以稍作改进.

算法4.17.

记号如定理 4.16 .

1. 设 $p_i = m_1 \cdots m_{i-1} \bmod m_i (i = 2, \cdots, n)$, 用扩展 Euclid 算法计算 u_i, v_i 使得 $u_i p_i + v_i m_i = 1$.

2. 顺次计算

$$\begin{aligned} y_1 &= x_1 \bmod m_1, \\ y_2 &= (x_2 - y_1)u_2 \bmod m_2, \\ &\vdots \\ y_i &= (x_i - (y_1 + m_1(y_2 + m_2(y_2 + \cdots m_{i-2}y_{i-1})\cdots)))u_i \bmod m_i. \end{aligned}$$

3. 输出 $x = y_1 + m_1(y_2 + m_2(y_3 + \cdots m_{n-1}y_n)\cdots)$.

算法本质上是求出 $x = \sum_{i=1}^{n} y_i p_i$ 中的系数 y_i, 用 p_i 代替原来的 M_i 减小了求逆的复杂度. 但由于需要第一步较复杂的预处理, 因此更适合在 m_i 给定的情况下进行多次求解的问题. 如果 m_i 总是变化的话, 将 n 个方程的方程组化为 $n-1$ 组两个方程的方程组更适合一些, 也就是说先求出满足 $x \equiv x_i \pmod{m_i}(i = 1, 2)$的解 x_{12}, 再求出满足

$$\begin{cases} x \equiv x_{12} \pmod{m_1 m_2}, \\ x \equiv x_3 \pmod{m_3}. \end{cases}$$

的解 x_{123} 等等, 如此递推即可.

4.6 连分数展式

任一实数的(简单)连分数展式可以按照定义逐项计算(倒数, 取整). 我们熟知数论中一个优美的结论: 一个数的连分数展式为循环的当且仅当其为二次无理数(二次整系数方程的根). 代数系统中尤其关心的也是二次无理数的展式可否有高效稳定的算法(例如应用在因子分解的 CFRAC 方法, Pell 方程的求解).

不失一般性, 设 $b_0 = \dfrac{\sqrt{d} + p_0}{q_0}$ 为二次无理数, 假设 $q_0 \mid d - p_0^2$(否则考虑 $\dfrac{\sqrt{dq_0^2} + p_0|q_0|}{q_0|q_0|}$)且有连分数展式

$$b_0 = a_0 + \frac{1}{b_1} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{b_2}} = \cdots = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}}},$$

则由定义得到

$$b_i = a_i + \frac{1}{b_{i+1}}.$$

设 $b_i = \dfrac{\sqrt{d}+p_i}{q_i}$, 则有

$$\frac{\sqrt{d}+p_i}{q_i} = a_i + \frac{q_{i+1}}{\sqrt{d}+p_{i+1}}.$$

比较有理部分和无理部分可知

$$\begin{cases} d + p_i p_{i+1} = a_i q_i q_{i+1} + q_i q_{i+1}, \\ p_i + p_{i+1} = a_i q_i. \end{cases} \tag{4.1}$$

将式 (4.1) 的第二式乘以 p_{i+1} 减去第一式得到

$$d - p_{i+1}^2 = q_i q_{i+1}. \tag{4.2}$$

在式 (4.2) 中以 i 代 $i+1$ 并作差得到

$$p_i^2 - p_{i+1}^2 = q_i(q_{i+1} - q_{i-1}).$$

再由 (4.1) 中第二式可知

$$a_i(p_i - p_{i+1}) = q_{i+1} - q_{i-1}.$$

最终得到

$$\begin{cases} a_i = \left\lfloor \dfrac{\sqrt{d}+p_i}{q_i} \right\rfloor, \\ p_{i+1} = a_i q_i - p_i, \\ q_{i+1} = a_i(p_i - p_{i+1}) + q_{i-1}. \end{cases}$$

其中约定 $q_{-1} = \dfrac{d-p_0^2}{q_0}$(利用 $\dfrac{d-p_0^2}{q_0}$ 为整数).

注79. 若 $q_i = 1$, 则 $a_i = \lfloor\sqrt{d}\rfloor + p_i$, 而若 $q_i > 1$, 不难验证 $a_i = \left\lfloor \dfrac{\lfloor\sqrt{d}\rfloor + p_i}{q_i} \right\rfloor$, 利用整数开方算法 4.4 可以只动用整数运算来递推计算展式中的 a_i.

4.7 素数计数函数 $\pi(x)$

很早人们就考虑计算小于等于 x 的素数个数 $\pi(x)$ 的问题, 并且数论中的一个基本问题便是研究 $\pi(x)$ 的性质(例如著名的素数定理). 古希腊人有一个直接的方

法: 用 Eratosthenes 筛将所有 x 以下的素数给找出来, 最后统计一下总数. 根据 Mertens 第二定理(参考 [119]),

$$\sum_{p\le x}\frac{1}{p}=\ln\ln x+a+O\left(\frac{1}{\ln x}\right),$$

其中 a 为常数, 求和针对素数 p. 因此这种完全筛法需要的操作次数为

$$\sum_{p\le\sqrt{x}}\frac{x}{p}=x\ln\ln\sqrt{x}+ax+O\left(\frac{x}{\ln\sqrt{x}}\right)=O(x\ln\ln x).$$

除了直接把所有素数求出还有没有其他办法呢? 可以使用组合的工具. 设 p_i 代表第 i 个素数, 根据容斥原理计算 x 以下的合数个数可以得到

$$\begin{aligned}1+\pi(x)-\pi(\sqrt{x})=\lfloor x\rfloor&-\sum_{p_i\le\sqrt{x}}\left\lfloor\frac{x}{p_i}\right\rfloor+\sum_{p_i<p_j\le\sqrt{x}}\left\lfloor\frac{x}{p_ip_j}\right\rfloor\\&-\sum_{p_i<p_j<p_k\le\sqrt{x}}\left\lfloor\frac{x}{p_ip_jp_k}\right\rfloor+\cdots.\end{aligned}$$

等式右端称为 Legendre 和. 然而 Lengendre 和中的非零项非常多(大约有 $\frac{6}{\pi}(1-\ln 2)x$ 项, 可参考 [110]), 并不适合直接应用于 $\pi(x)$ 的计算. 为了减少 Legendre 和中的项数, 19 世纪德国天文学家 Meissel 提出了部分筛法并正确求得了 $\pi(10^8)$. 1959 年, Lehmer 改进了 Meissel 的方法, 并用计算机求得了 $\pi(10^{10})$. 1985 年, Lagarias, Miller, Odlyzko[110] 用更精细的 Meissel-Lehmer 方法求得了 $\pi(4\times10^{16})$, 1996 年, Deléglise 和 Rivat[65] 证明了可以在 $O\left(\frac{x^{2/3}}{\ln^2 x}\right)$ 时间和 $O(x^{1/3}\ln^3 x\ln\ln x)$ 空间内计算 $\pi(x)$, 并将计算记录提高到了 $\pi(10^{18})$. 目前的世界纪录为分布式计算项目pi(x)[1]保持的 $\pi(4\times10^{22})$.

4.7.1 部分筛函数

部分筛函数定义为

$$\phi(x,a):=\#\{\text{正整数 }n\le x\mid n\text{ 的所有素因子}>p_a\},$$

再设

$$P_k(x,a):=\#\{\text{正整数 }n\le x\mid n\text{ 恰有 }k\text{ 个素因子且所有素因子}>p_a\}.$$

[1]http://numbers.computation.free.fr/Constants/Primes/Pix/pixproject.html

约定 $P_0(x,a) := 1$, 根据定义可以得到

$$\phi(x,a) = \sum_{k=0}^{\infty} P_k(x,a),$$

其中的求和实际上是有限和, 因为当 $k \geq \log_{p_a} x$ 时 $P_k(x,a)$ 恒为零. 为了减少求和的项数, 我们取 y 满足 $x^{1/3} \leq y \leq x^{1/2}$, $a = \pi(y)$, 则求和只有三项非零:

$$\begin{aligned}\phi(x,a) &= P_0(x,a) + P_1(x,a) + P_2(x,a)\\ &= 1 + (\pi(x) - a) + P_2(x,a).\end{aligned}$$

从而

$$\pi(x) = \phi(x,a) + P_2(x,a) + a - 1,$$

我们只需要高效地求出 $\phi(x,a)$ 与 $P_2(x,a)$ 即可.

4.7.2 计算 $P_2(x,a)$

由定义(总假定 n 为正整数),

$$P_2(x,a) = \#\{n \leq x \mid n = pq, \text{素数 } p,q > p_a\},$$

故有

$$\begin{aligned}P_2(x,a) &= \sum_{y<p\leq\sqrt{x}} \#\{\text{素数 } q \mid p \leq q \leq \frac{x}{p}\}\\ &= \sum_{y<p\leq\sqrt{x}} \pi\left(\frac{x}{p}\right) - \pi(p) + 1.\end{aligned}$$

由于 $\dfrac{x}{p} \leq x^{2/3}$, $p \leq x^{1/2}$, 因此可用直接用 $[1, x^{2/3}]$ 上的完全筛法在 $O(x^{2/3+\varepsilon})$ 时间内求得 $P_2(x,a)$.

4.7.3 计算 $\phi(x,a)$

部分筛函数最重要的组合性质之一是如下的等式

$$\phi(x,a-1) = \phi(x,a) + \phi(\frac{x}{p_a}, a-1),$$

右边第一项的组合意义为 $\phi(x,a-1)$ 对应的数中不含有因子 p_a 的那部分, 而第二项则表示含有因子 p_a 的那部分. 从而可得递推关系

$$\begin{cases}\phi(x,a) = \phi(x,a-1) - \phi(\dfrac{x}{p_a}, a-1),\\ \phi(x,0) = \lfloor x \rfloor.\end{cases}$$

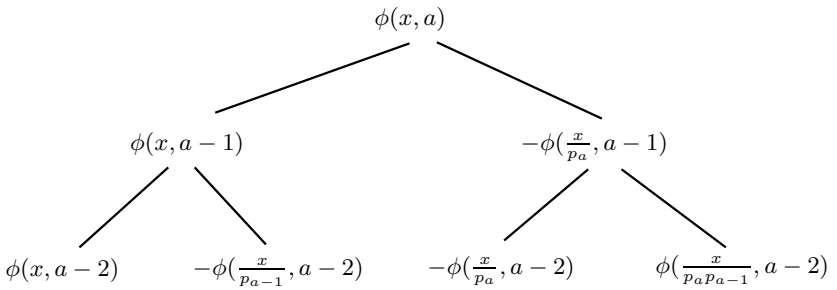

图 4.2: 计算 $\phi(x,a)$ 的二叉树

由此可画出 $\phi(x,a)$ 的二叉树(仅画出前三层), 递归到二叉树的叶结点最终得到

$$\phi(x,a)=\sum_{\substack{n\le x\\ P^+(n)\le y}}\mu(n)\left\lfloor\frac{x}{n}\right\rfloor, \tag{4.3}$$

其中 $\mu(n)$ 为 Möbius 函数, $P^+(n)$ 表示 n 的最大素因子.

但 (4.3) 中的求和项仍然太多, 例如有三个素因子均 $\le y$ 的数的个数为 $O((x^{1/3}/\ln x^{1/3})^3)=O(x/\ln^3 x)$. 因此我们需要对二叉树进行剪枝, 例如可以采取如下的剪枝规则:

1. $a=0$ 且 $n\le y$; (叶结点)
2. $n>y$. (层数过深)

由如上的剪枝规则可得 $\phi(x,a)=S_0+S$, 其中

$$S_0=\sum_{n\le y}\mu(n)\left\lfloor\frac{x}{n}\right\rfloor,$$

$$S=\sum_{n/P^-(n)\le y<n}\mu(n)\phi\left(\frac{x}{n},\pi(P^-(n))-1\right), \tag{4.4}$$

$P^-(n)$ 表示 n 的最小素因子.

由于 $n\le x^{1/2}$, $\dfrac{x}{n}\le x^{2/3}$, 可以在 $O(x^{2/3})$ 的时间内计算 S_0.

4.7.4 计算 S

设 $p=P^-(n)$, $n=mp$, 则 $\mu(n)=-\mu(m)$ 且 $P^-(m)>p$, 式 (4.4) 变为

$$S=-\sum_{p\le y}\sum_{\substack{P^-(m)>p\\ m\le y<mp}}\mu(m)\phi\left(\frac{x}{mp},\pi(p)-1\right).$$

可将 S 的外层求和拆为三项 $S_1+S_2+S_3$, 求和分别对 $x^{1/3}<p\le y$, $x^{1/4}<p\le x^{1/3}$ 及 $p\le x^{1/4}$. 注意到当 $p>x^{1/4}$ 时, 若 m 非素数, 由 $P^-(m)>p$ 可知 $m>p^2>x^{1/2}\ge y$, 从而内层的求和为零, 因此对 S_1 和 S_2 可以考虑 m 为素数 q 的情形, 且此时 $\mu(m)=-1$.

4.7.5 计算 S_1

我们需要计算

$$S_1=\sum_{x^{1/3}<p\le y}\sum_{p<q\le y}\phi\left(\frac{x}{pq},\pi(p)-1\right).$$

注意到当 $q>p>x^{1/3}$ 时, $\dfrac{x}{pq}<x^{1/3}<p$, 故 $\phi\left(\dfrac{x}{pq},\pi(p)-1\right)=1$. 由 $x^{1/3}<p<q\le y$ 可知

$$S_1=\binom{\pi(y)-\pi(x^{1/3})}{2},$$

可直接计算.

4.7.6 计算 S_3

我们需要计算

$$S_3=-\sum_{p\le x^{1/4}}\sum_{\substack{P^-(m)>p\\ m\le y<mp}}\mu(m)\phi\left(\frac{x}{mp},\pi(p)-1\right).$$

可以如下进行: 首先用 $x^{1/4}$ 以下的素数对 $[1,\frac{x}{y}]$ 进行一次筛法, 每次用一个素数 p_k 筛选过后, 对所有 $[\frac{y}{p},y]$ 中的无平方因子且还没有被之前素数筛出的的 m 累加 $\mu(m)\phi\left(\frac{x}{mp},\pi(p)-1\right)$(这些值可在前面量的计算过程中进行保存). 由于 $m\le y$, $p\le x^{1/4}$, 计数过程的时间复杂度为 $O(yx^{1/4})$.

4.7.7 计算 S_2

我们需要计算

$$S_2=\sum_{x^{1/4}<p\le x^{1/3}}\sum_{p<q\le y}\phi\left(\frac{x}{pq},\pi(p)-1\right).$$

注意到当 $q>\dfrac{x}{p^2}$ 时, 有 $\dfrac{x}{pq}<p$, 从而 $\phi(\frac{x}{pq},\pi(p)-1)$ 恒为 1. 从而可将内层的求和

分为 $q > \dfrac{x}{p^2}$ 与 $q \le \dfrac{x}{p^2}$ 两段, 记为 $S_2 = U + V$. 立即可知

$$\begin{aligned} U &= \sum_{x^{1/4} < p \le x^{1/3}} \#\{\text{素数 } q \mid \frac{x}{p^2} < q \le y\} \\ &= \sum_{x^{1/4} < p \le x^{1/3}} \pi(y) - \pi(\frac{x}{p^2}). \end{aligned}$$

由于 $\dfrac{x}{p^2}, y \le \sqrt{x}$, 因此可以通过完全筛法在 $O(x^{1/2+\varepsilon})$ 时间内求得 U.

4.7.8 计算 V

此时有 $q \le \dfrac{x}{p^2}$, 即 $p \le \dfrac{x}{pq} < x^{1/2} < p^2$, 从而

$$\begin{aligned} \phi\left(\frac{x}{pq}, \pi(p)-1\right) &= P_0\left(\frac{x}{pq}, \pi(p)-1\right) + P_1\left(\frac{x}{pq}, \pi(p)-1\right) \\ &= 1 + \pi\left(\frac{x}{pq}\right) - (\pi(p)-1) \\ &= 2 - \pi(p) + \pi\left(\frac{x}{pq}\right). \end{aligned}$$

因此有

$$\begin{aligned} V &= \sum_{x^{1/4} < p \le x^{1/3}} \sum_{\substack{p < q \le y \\ q \le x/p^2}} (2 - \pi(p)) + \sum_{x^{1/4} < p \le x^{1/3}} \sum_{\substack{p < q \le y \\ q \le x/p^2}} \pi\left(\frac{x}{pq}\right) \\ &=: V_1 + V_2. \end{aligned}$$

由于 $p \le x^{1/3}$, V_1 可以通过之前的筛法结果直接求得.

4.7.9 计算 V_2

我们需要计算

$$V_2 = \sum_{x^{1/4} < p \le x^{1/3}} \sum_{\substack{p < q \le y \\ q \le x/p^2}} \pi\left(\frac{x}{pq}\right).$$

这里求和的项数非常之多, 因此也是算法中最复杂的部分. 首先简化关于 q 的两个限制, 改写为

$$V_2 = \sum_{x^{1/4} < p < \sqrt{\frac{x}{y}}} \sum_{p < q \le y} \pi\left(\frac{x}{pq}\right) + \sum_{\sqrt{\frac{x}{y}} < p < x^{1/3}} \sum_{p < q \le y} \pi\left(\frac{x}{pq}\right).$$

将内外层求和分为五段, 分别记为(省去求和项)

$$W_1 = \sum_{x^{1/4}<p\le\frac{x}{y^2}} \sum_{p<q\le y},$$

$$W_2 = \sum_{\frac{x}{y^2}<p\le\sqrt{\frac{x}{y}}} \sum_{p<q\le\sqrt{\frac{x}{p}}},$$

$$W_3 = \sum_{\frac{x}{y^2}<p\le\sqrt{\frac{x}{y}}} \sum_{\sqrt{\frac{x}{p}}<q\le y},$$

$$W_4 = \sum_{\frac{x}{y^2}<p\le x^{1/3}} \sum_{p<q\le\sqrt{\frac{x}{p}}},$$

$$W_5 = \sum_{\frac{x}{y^2}<p\le x^{1/3}} \sum_{\sqrt{\frac{x}{p}}<q\le\frac{x}{p^2}}.$$

W_1, W_2 中的求和项满足 $y < \frac{x}{pq} < x^{1/2}$, 可以通过的筛法结果进行计算, 而 W_3, W_4, W_5 只能"老老实实"地通过求和进行计算了. 由于 W_3 与 W_5 中当固定 p 遍历时 $\pi\left(\frac{x}{pq}\right)$ 变化不大, 因此可以做一个略微的改进: 通过查表来确定下一个 $\pi\left(\frac{x}{pq}\right)$ 值改变的 q, 从而减小计算量. 至此我们已经得到了计算 $\pi(x)$ 的完整算法.

注80. 1987 年 Lagarias 和 Odlyzko[111] 提出了完全不同的分析方法来计算 $\pi(x)$, 方法基于对 Riemann ζ 函数的积分变换的数值积分, 其中一个版本的算法需要 $O(x^{3/5+\varepsilon})$ 时间与 $O(x^{\varepsilon})$ 空间, 另一个版本的算法需要 $O(x^{1/2+\varepsilon})$ 时间与 $O(x^{1/4+\varepsilon})$ 空间. 但目前为止还没有被真正实现, 并且估计在 10^{17} 以下还是要比 Lagarias-Miller-Odlyzko 方法慢.

注81. Gourdon[78] 在 2001 年提出了一个新的改进方法(主要是针对 V_2 的计算), 并给出了可行的并行化算法.

4.8 第 n 个素数 p_n

相对对给定的数进行素性判定来说, 给出第 n 个素数是更为困难的事情, 对于很大的 n 来说, 纯粹使用素性检测挨个找素数, 或者利用筛法找出所有素数, 都不切实际. 不过我们已经知道 $\pi(n)$ 如何计算, 而 p_n 在某种意义上是 $\pi(n)$ 的"反函数"($\pi(p_n) = n$, $p_{\pi(n)} \le n$ 且当 n 为素数时等号成立), 我们利用这一点可以反过来计算 p_n.

Lehmer[113] 提出了一个一般的求此类"逆函数"的算法, 他考虑迭代

$$S_k = S_{k-1} + n - \pi(S_{k-1}),$$

取适当的初值 $S_0 < \pi(n)$, 容易看出 S_k 严格单调递增, 直到 $S_k = \pi(n)$ 后恒为常数. 然而此迭代过程收敛过慢, 例如求 $n = 1000$ 时的 $p_n = 7919$, 取 $S_0 = 7000$ 仍需要 49 步迭代.

我们可以考虑二分搜索的办法, 估计一个区间范围 $[a, b]$ 使得 p_n 落在其中, (可通过计算 $\pi(a) \le n \le \pi(b)$ 来确证). 然后二分区间 $[a, b]$ 为 $[a, \lfloor \frac{a+b}{2} \rfloor]$, $[\lfloor \frac{a+b}{2} \rfloor, b]$, 并计算 $\pi(\lfloor \frac{a+b}{2} \rfloor)$ 判断 p_n 落在哪一个子区间中, 如此续行, 最终搜索出 p_n 的位置.

4.9 Möbius 函数 $\mu(n)$ 和 Euler 函数 $\varphi(n)$

Möbius 函数 $\mu(n)$和 Euler 函数 $\varphi(n)$均为重要的数论函数. Möbius 函数定义为

$$\mu(n) = \begin{cases} 0 & \text{若 } n \text{ 有平方因子,} \\ 1 & \text{若 } n = 1, \\ (-1)^k & \text{若 } n \text{ 为 } k \text{ 个不同素因子的乘积.} \end{cases}$$

Euler 函数定义为

$$\varphi(n) = n \prod_{p|n} (1 - \frac{1}{p}).$$

可以用类似于计算 $\pi(x)$ 的方法计算 $M(n) = \sum_{k=1}^{n} \mu(k)$, 从而反求出 $\mu(n)$(参见 [66]). 这种组合方法的时间复杂度为 $O(x^{2/3+\varepsilon})$, 但实践中未必比直接对 n 分解根据定义计算更优, 因为就算 n 很大(例如 $n > 10^{20}$), 往往也有机会可以快速地分解, 然而组合方法则由于规模太大而无法使用.

对于 Euler 函数, 我们熟知理论上重要的反演公式

$$\varphi(n) = \sum_{d|n} \mu(d) \cdot \frac{n}{d},$$

但同样无法直接用于 $\varphi(n)$ 的计算. 实践中宜直接采取分解并根据定义计算的方法.

5 数学常数

数学常数凝集着数学研究的精华, 计算任意精度的数学常数是符号计算中一项重要的内容, 许多涉及到微积分, 特殊函数和函数求值的运算都会涉及到数学常数. 特别地, 如果运算结果关于这些数学常数是非线性的, 那么数学常数的精度就显得尤为重要, 下面主要以圆周率为例介绍一下计算数学常数时用到的主要方法.

5.1 圆周率

人们对于计算圆周率π 的兴趣似乎从远古时代就开始了, 如果只从较为系统的研究看起, 刘徽的割圆术大概是微积分被发现之前用来计算 π 的最佳方法, 祖冲之利用它求出了著名的密率 $\frac{355}{113}$, 一共有 8 位有效数字.1706 年 Taylor 的私人教师 Machin 利用反正切函数的 Taylor 级数展开式借助纸笔演算求出了 π 的小数点后 100 位数字, 1949 年美国使用早期的电子计算机算出 π 的小数点后的 2037 位数, 目前计算 π 的小数位数的世界纪录(参见 [98])保持者是日本科学家 Yasumasa Kanada(金田康正), 他于 2002 年在一台超级并行计算机上将 π 的小数位数推进到了惊人的 12411 亿位.

5.1.1 级数方法

折半求和

折半求和(Binary Splitting)(参见 [55])是用来计算超几何级数(参见 [1])在有理点处的值的一项重要方法.

超几何级数是一类特殊的幂级数.

定义5.1 (升阶乘)**.** α 的 n 阶升阶乘定义为

$$(\alpha)_n = \frac{\Gamma(\alpha+n)}{\Gamma(\alpha)} = \alpha(\alpha+1)\cdots(\alpha+n-1).$$

定义5.2 (超几何级数)**.** 称形如

$${}_pF_q(a_1,\cdots,a_p;b_1,\cdots,b_q;z) = \sum_{n=0}^{\infty} \frac{(a_1)_n\cdots(a_p)_n z^n}{(b_1)_n\cdots(b_q)_n n!}$$

的级数为超几何级数.

注82. 许多函数都具有超几何级数的展开式, 最简单的例子是 $e^z = {}_0F_0(z)$. 除此之外, 初等函数一般都具有 ${}_2F_1$ 形式的展开式.

许多数学常数可以看作超几何级数在某个有理点处的值. 一般来说, 超几何级数都可以改写成如下形式

$$S = \sum_{n=0}^{\infty} \frac{a(n)}{b(n)} \frac{p(0)\cdots p(n)}{q(0)\cdots q(n)},$$

其中 $a(n), b(n), p(n), q(n)$ 都是整系数多项式.

例5.1. 以 Chudnovsky 公式为例, 先将通项改写成

$$s(n) = \frac{(-1)^n(An+B)}{1} \cdot \frac{(6n)!/(3n)!}{(n!)^3 C^{3n}},$$

于是

$$\begin{aligned} a(n) &= (-1)^n(An+B), \\ b(n) &= 1, \\ p(0)\cdots p(n) &= \frac{(6n)!}{(3n)!}, \\ q(0)\cdots q(n) &= (n!)^3 C^{3n}. \end{aligned}$$

相邻两项相除就可以解出 $p(n) = (6n-1)(6n-3)(6n-5)$, $q(n) = n^3C^3$.

考虑部分和

$$S = S(n_1, n_2) = \sum_{n_1 \le n < n_2} \frac{a(n)}{b(n)} \frac{p(n_1)\cdots p(n)}{q(n_1)\cdots q(n)}.$$

记

$$P = P(n_1, n_2) = p(n_1) \cdots p(n_2 - 1),$$
$$Q = Q(n_1, n_2) = q(n_1) \cdots q(n_2 - 1),$$
$$B = B(n_1, n_2) = b(n_1) \cdots b(n_2 - 1),$$

再设 $T = T(n_1, n_2) = BQS$, 那么 $S = \frac{T}{BQ}$.

折半求和使用了一种类似于通分的方法构造出递推公式来计算 P, Q, B, T.

定理5.1. *设 n_m 满足 $n_1 < n_m < n_2$, 将区间 $n_1 \le x < n_2$ 分成两段 $n_1 \le x < n_m$ 及 $n_m \le x < n_2$, 分别用下标 l, r 来表示对应这两个子区间的 P, Q, B, T 的值, 例如 $P_l = P(n_1, n_m), P_r = P(n_m, n_2)$, 那么存在递推公式*

$$P = P_l \cdot P_r,$$
$$Q = Q_l \cdot Q_r,$$
$$B = B_l \cdot B_r,$$
$$T = T_l \cdot B_r Q_r + T_r \cdot B_l P_l.$$

现在可以写出折半求和算法的详细过程了.

算法5.1 (折半求和)**.**

输入:通项公式 $a(n), b(n), p(n), q(n)$, 区间 $[n_1, n_2)$.

输出:部分和 $P(n_1, n_2), Q(n_1, n_2), B(n_1, n_2), T(n_1, n_2)$.

1. 如果 $n_2 - n_1 < 5$, 直接根据定义计算出 (P, Q, B, T) 并返回.
2. 设 $n_m = \left[\frac{n_1+n_2}{2}\right]$, (P_l, Q_l, B_l, T_l), (P_r, Q_r, B_r, T_r) 分别是对应于子区间 $[n_1, n_m)$ 和 $[n_m, n_2)$ 上的 (P, Q, B, T), 在两个子区间上分别应用折半求和计算出它们.
3. 计算 $P = P_l P_r$, $Q = Q_l Q_r$, $B = B_l B_r$ 和 $T = T_l B_r Q_r + T_r B_l P_l$, 返回 (P, Q, B, T).

注83. 使用折半求和计算出 $[n_1, n_2)$ 上的部分和 (P, Q, B, T) 后, 再利用 $S = \frac{T}{BQ}$ 做一次除法就可以求出原级数的部分和 S.

注意到最终结果 S 只依赖于 T 和 BQ 的比值, 因此在折半求和计算 (P, Q, B, T) 的过程中, 最后一步应用递推公式之前可以将 P_l, Q_r 先"约分", 即将它们同时除

以其最大公因子 $\gcd(P_l, Q_r)$, 这样一来由递推公式求出的 P, Q, T 都变为原来的 $\frac{1}{\gcd(P_l,Q_r)}$ 倍, 而比值 $\frac{T}{BQ}$ 则保持不变.

为了快速地计算出 $\gcd(P_l, Q_r)$, 常用的方法是在计算过程中保留 P, Q 的部分素因子分解式, 并在递推相乘时同步更新其部分素因子分解式.

从整体上来看, 折半求和并没有减少运算次数, 它只是让参与乘法的两个数的数量级更接近, 这样就可以充分发挥 Karatsuba, Toom-Cook 和 FFT 等大整数快速乘法算法的优势. 如果大整数乘法采用古典乘法算法, 那么折半求和是不起作用的, 甚至可能比普通的级数求和方法更慢. 除此之外, 折半求和实际上是将截断的级数先乘上分母的最小公倍数后再计算, 这样可以避免对中间结果进行除法等不精确计算, 而只需要在最后做一次除法. 可以证明, 采用折半求和算法计算出部分和 S 的前 N 位有效数字大约需要 $O((\log N)^2 M(N))$ 次基本运算, 其中 $M(N)$ 代表两个 N 位整数相乘所需要的基本运算的次数(如果采用有限域上的 FFT 乘法, 那么 $M(N) = O(N \log N \log\log N)$). 折半求和利用两个独立的子区间的数据来合成整个区间的数据, 因此也很适合于并行计算.

Machin 型公式

John Machin 用来计算 π 的公式是

定理5.2 (Machin 公式)**.**

$$\frac{\pi}{4} = 4\tan^{-1}\frac{1}{5} - \tan^{-1}\frac{1}{239}.$$

证明. 考虑恒等式 $\tan(\frac{\pi}{4} - x) = \frac{1-\tan x}{1+\tan x}$, 两边同时取反正切得到

$$\frac{\pi}{4} = x + \tan^{-1}\frac{1-\tan x}{1+\tan x},$$

再令 $x = 4\tan^{-1}\frac{1}{5}$ 就可以证明 Machin 公式. □

在 Machin 公式(定理 5.2)中利用 $\tan^{-1} x$ 的 Taylor 展开式

$$\tan^{-1} x = x - \frac{x^3}{3} + \frac{x^5}{5} + \cdots + (-1)^n \frac{x^{2n+1}}{2n+1} + O(x^{2n+3}),$$

可以得到

定理5.3.

$$\frac{\pi}{4} = 4\sum_{k=0}^{\infty}\frac{(-1)^k}{(2k+1)5^{2k+1}} - \sum_{k=0}^{\infty}\frac{(-1)^k}{(2k+1)239^{2k+1}}.$$

注84. 这个公式的优点是第二项收敛得很快, 而第一项的分母中含有 5 的方幂, 便于手工在十进制下计算.

定义5.3 (Machin 型公式)**.** 形如

$$\frac{\pi}{4}=\sum_{k=0}^{n-1}a_k\tan^{-1}\frac{1}{b_k},\quad a_k,b_k\in\mathbb{N}_+,$$

的公式, 或者写成复数形式

$$i=\prod_{k=0}^{n-1}\left(\frac{b_k+i}{b_k-i}\right)^{a_k},\quad a_k,b_k\in\mathbb{N}_+,$$

统称为 Machin 型公式或 n 阶 Machin 型公式.

例5.2. 以 $n=2$ 为例, 其他三个 2 阶 Machin 型公式是

$$\begin{aligned}\frac{\pi}{4}&=\tan^{-1}\frac{1}{2}+\tan^{-1}\frac{1}{3}\\&=2\tan^{-1}\frac{1}{2}-\tan^{-1}\frac{1}{7}\\&=2\tan^{-1}\frac{1}{3}+\tan^{-1}\frac{1}{7}.\end{aligned}$$

将 Machin 型公式(定义 5.3)的复数形式展开, 并注意到 $a_k,b_k\in\mathbb{N}_+$, ,我们就可以将寻找 Machin 型公式的问题转化成寻找高阶不定方程整数解的问题. 据此还可以证明 $n=2$ 的 Machin 型公式一共只有以上四种, 而 n 从 1 到 21 所有的 Machin 型公式共有 1500 种(参见 [182]).

定义5.4 (Lehmer 数)**.** Machin 型公式

$$\frac{\pi}{4}=\sum_{k=0}^{n-1}a_k\tan^{-1}\frac{1}{b_k},\quad a_k,b_k\in\mathbb{N}_+$$

的 Lehmer 数(参见 [115])为

$$\sum_{k=0}^{n-1}\frac{1}{\log_{10}b_k}$$

注85. 不同 Machin 型公式的收敛速度可以用 Lehmer 数来衡量, Lehmer 数越小, 公式的收敛速度越快.

例5.3. 目前已知收敛最快的 Machin 型公式是由黄见利(参见 [90])发现的

$$\begin{aligned}\frac{\pi}{4}&=183\tan^{-1}\frac{1}{239}+32\tan^{-1}\frac{1}{1023}-68\tan^{-1}\frac{1}{5832}\\&\quad+12\tan^{-1}\frac{1}{110443}-12\tan^{-1}\frac{1}{4841182}-100\tan^{-1}\frac{1}{6826318},\end{aligned}$$

它对应的 Lehmer 数是 1.51244.

与其他的计算方法相比, Machin 型公式更适合于并行计算. 目前计算 π 的位数的世界纪录是 2002 年 12 月 6 日由东京大学信息基础中心超级计算研究部门的 Yasumasa Kanada(金田康正)教授正式发表的, 他利用日本日立公司(HITACHI)制作提供的超级计算机“SR8000/MPP”花了大约 600 个小时进行计算和验算, 计算时采用了如下两个 $n=4$ 的 Machin 型公式

$$\begin{aligned}\frac{\pi}{4} &= 12\tan^{-1}\frac{1}{49}+32\tan^{-1}\frac{1}{57}-5\tan^{-1}\frac{1}{239}+12\tan^{-1}\frac{1}{110443}\\ &= 44\tan^{-1}\frac{1}{57}+7\tan^{-1}\frac{1}{239}-12\tan^{-1}\frac{1}{682}+24\tan^{-1}\frac{1}{12943}.\end{aligned}$$

Ramanujan 型公式

Ramanujan 构造过一个 $\frac{1}{\pi}$ 的超几何级数展开式(参见 [29]).

定理5.4.

$$\frac{1}{\pi}=\frac{2\sqrt{2}}{9801}\sum_{n=0}^{\infty}\frac{(4n)!}{(n!)^4}\cdot\frac{1103+26390n}{396^{4n}}.$$

注86. 后来 Chudnovsky 兄弟(参见 [54])和 Borwein 兄弟(参见 [34])又分别给出形如

$$\frac{1}{\pi}=12\sum_{n=0}^{\infty}\frac{(-1)^n(6n)!}{(3n)!(n!)^3}\cdot\frac{An+B}{C^{3n+\frac{3}{2}}}$$

的三组公式, 这三组公式的参数分别为 $A=545140134, B=13591409, C=640320$,

$$\begin{aligned}A &= 1657145277365+212175710912\sqrt{61},\\ B &= 107578229802750+13773980892672\sqrt{61},\\ C &= 5280(236674+30303\sqrt{61})\end{aligned}$$

和

$$
\begin{aligned}
A = {} & 633650283129719995854262 20 \\
& + 28337702140800842046825600\sqrt{5} \\
& + 384\sqrt{5}(10891728551171178200467436212395209160385656017 \\
& \quad + 48709290865788102250773385345416887213512550 40\sqrt{5})^{1/2}, \\
B = {} & 784991045349662721028974900 0 \\
& + 35105866782609320289656064 00\sqrt{5} \\
& + 2515968\sqrt{3110}(62602083237890016369933226544440208821 61 \\
& \quad + 27996502730604442965772068907188251902 35\sqrt{5})^{1/2}, \\
C = {} & -214772995063512240 \\
& - 96049403338648032\sqrt{5} \\
& - 1296\sqrt{5}(10985234579463550323713318473 \\
& \quad + 49127462536923627546073959 12\sqrt{5})^{1/2}.
\end{aligned}
$$

注87. 以上四个公式统称为 Ramanujan 型公式, 利用代数数论和二次域的知识还可以构造出更多这样的 Ramanujan 型公式(参见 [183]), 不同 Ramanujan 型公式的收敛速度可以用每计算一个级数项后结果所增加的十进制有效位数来衡量, 以上这四个公式每向后计算一项, 结果的有效位数分别大约增加 8 位, 14 位, 31 位和 50 位(参见 [183]).

在个人计算机上计算 π 时, Ramanujan 型公式是最常用的级数展开式, 它可以利用折半求和的方法来计算(参见算法 5.1). 目前个人计算机上 π 的小数位数世界纪录的保持者是由 Steve Pagliarulo 开发的 QPI[133], 它仅需数秒就能求出 π 的前 100 万位, 计算时采用的就是第二个 Ramanujan 型公式, 它是由 Chudnovsky 兄弟发现的, 因此常常也被称为 Chudnovsky 公式.

BBP 公式

最后值得一提的是 Borwein 兄弟和其合作者共同发现的 BBP 公式(参见 [22]).

定理5.5.

$$
\pi = \sum_{k=0}^{\infty} \frac{1}{16^k}\left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6}\right).
$$

注88. 注意到通项前的系数是 $\frac{1}{16^k}$, 因此可以利用它直接求出 π 在十六进制下某一个指定位上的数字, 而不必先求出所有之前位上的数字.

在此基础上 F. Bellard 又提出了一个公式(参见 [26]).

定理5.6.

$$\pi = \frac{1}{2^6}\sum_{n=0}^{\infty}\frac{(-1)^n}{2^{10n}}(-\frac{2^5}{4n+1} - \frac{1}{4n+3} + \frac{2^8}{10n+1} - \frac{2^6}{10n+3} - \frac{2^2}{10n+5} - \frac{2^2}{10n+7} + \frac{1}{10n+9}).$$

注89. 用它计算要比 BBP 公式快 40

5.1.2 迭代方法

顾名思义, 迭代方法就是利用递推公式来计算 π, 通过迭代逐步提高精度.

将递推公式设法改写成 $\pi_{n+1} = f(\pi_n)$ 的形式, 则存在正实数 m 使得 $|\pi_{n+1} - \pi| = |f(\pi_n) - \pi| \sim |\pi_n - \pi|^m$, 于是每迭代一步后的误差将变为原来误差的 m 次幂, 如果换一个角度来看, 这就是说有效位数将变为原来有效位数的 m 倍, 因此可以用 m 的大小作为衡量迭代法性能的指标.

代数几何平均值

首先来看 Brent-Salamin 算法(参见 [38]), 这是一个二阶收敛的迭代算法, 它用到了 Gauss-Legendre 的代数几何平均值(AGM)迭代(参见 [33]).

定义5.5 (代数几何平均值)**.** 设 $a, b \in \mathbb{R}^+$, $a > b$. 令 $a_0 = a$, $b_0 = b$, $a_{n+1} = \frac{1}{2}(a_n + b_n)$, $b_{n+1} = \sqrt{a_n b_n}$, 则 a, b 的代数几何平均值定义为

$$\mathrm{AGM}(a, b) = \lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n.$$

注90. 如果将代数平均值或几何平均值换成更高阶的平均值, 例如

$$(\frac{a_n^3 b_n + b_n^3 a_n}{2})^{1/4},$$

还可以得到更高阶的迭代算法.

完全椭圆积分

为了利用代数几何平均值来计算圆周率, 还需要用到完全椭圆积分的概念.

定义5.6 (完全椭圆积分)**.** 设 $R(x,y)$ 是有理函数, 其中 $y^2=P(x)$ 是 x 的三次或四次多项式, 那么称形如 $\int R(x,y)dx$ 积分为椭圆积分. 特别地, 如下两个积分

$$K(k)=\int_0^1\frac{dx}{\sqrt{(1-x^2)(1-k^2x^2)}}=\int_0^{\frac{\pi}{2}}\frac{d\phi}{\sqrt{1-k^2sin^2\phi}},$$
$$E(k)=\int_0^1\sqrt{\frac{1-k^2x^2}{1-x^2}}\,dx=\int_0^{\frac{\pi}{2}}\sqrt{1-k^2sin^2\phi}\,d\phi,$$

分别称为第一类和第二类完全椭圆积分, 其中 $|k|<1$.

定义5.7.

$$I(a,b)=\frac{1}{a}K(\sqrt{1-\frac{b^2}{a^2}}).$$

定理5.7.

$$I(a,b)=\frac{\pi}{2\mathrm{AGM}(a,b)}.$$

证明. 根据第一类完全椭圆积分的定义(定义 5.6)我们有

$$\begin{aligned}I(a,b)&=\int_0^{\frac{\pi}{2}}\frac{d\phi}{\sqrt{a^2-(a^2-b^2)\sin^2\phi}}\\&=\int_0^{\frac{\pi}{2}}\frac{d\phi}{\sqrt{a^2\cos^2\phi+b^2\sin^2\phi}}.\end{aligned}$$

通过变量替换

$$u=\frac{1}{2}(t-\frac{ab}{t})$$

可以得出 $I(a,b)=I(\frac{a+b}{2},\sqrt{ab})$.

故 $I(a_0,b_0)=I(a_n,b_n)=I(a_{n+1},b_{n+1})$, 两边取极限得到

$$\begin{aligned}I(a,b)&=I(\mathrm{AGM}(a,b),\mathrm{AGM}(a,b))\\&=\frac{1}{\mathrm{AGM}(a,b)}K(0)=\frac{\pi}{2\mathrm{AGM}(a,b)}.\end{aligned}$$

□

注91. 注意到 $I(1,\frac{1}{\sqrt{2}})=K(\frac{1}{\sqrt{2}})$, 因此

$$K(\frac{1}{\sqrt{2}})=\frac{\pi}{2\mathrm{AGM}(1,\frac{1}{\sqrt{2}})}.$$

定义5.8.

$$J(a,b)=aE\left(\sqrt{1-\frac{b^2}{a^2}}\right).$$

定理5.8.

$$E(\frac{1}{\sqrt{2}}) = K(\frac{1}{\sqrt{2}})(1 - \sum_{n=0}^{\infty} 2^{n-1}c_n^2).$$

证明. 根据第二类完全椭圆积分的定义(定义 5.6)我们有

$$J(a,b) = \int_0^{\frac{\pi}{2}} \sqrt{a^2\cos^2\phi + b^2\sin^2\phi}\, d\phi.$$

令 $a = 1, b = \cos\phi$, 于是 $K(\sin\phi) = \frac{\pi}{2\mathrm{AGM}(a,b)}$, 再令 $c_0 = \sin\phi, c_{n+1} = a_n - a_{n+1}$, 可以证明

$$\sum_{n=0}^{\infty} 2^{n-1}c_n^2 = 1 - \frac{E(\sin\phi)}{K(\sin\phi)}.$$

因此

$$E(\frac{1}{\sqrt{2}}) = K(\frac{1}{\sqrt{2}})(1 - \sum_{n=0}^{\infty} 2^{n-1}c_n^2).$$

□

定理5.9.

$$\pi = \frac{(2\mathrm{AGM}(1, \frac{1}{\sqrt{2}})^2}{2 - 4\sum_{n=0}^{\infty} 2^{n-1}c_n^2}.$$

证明. 若 $\phi + \psi = \frac{\pi}{2}$, 那么第一类与第二类完全椭圆积分之间存在 Legendre 关系

$$K(\sin\phi)E(\sin\psi) + E(\sin\phi)K(\sin\psi) - K(\sin\phi)K(\sin\psi) = \frac{\pi}{2}.$$

令 $\phi = \psi = \frac{\pi}{4}$ 就可以得到 $\pi = 4K(\frac{1}{\sqrt{2}})E(\frac{1}{\sqrt{2}}) - 2K(\frac{1}{\sqrt{2}})^2$, 即

$$\pi = K^2(2 - 4\sum_{n=0}^{\infty} 2^{n-1}c_n^2),$$

也可以写成

$$\pi = \frac{(2\mathrm{AGM}(1, \frac{1}{\sqrt{2}})^2}{2 - 4\sum_{n=0}^{\infty} 2^{n-1}c_n^2}.$$

□

Brent-Salamin 算法

根据定理 5.9 , 我们现在可以写出 Brent-Salamin 算法的详细过程了.

算法5.2 (Brent-Salamin 算法).

输入:迭代次数 m.

输出:迭代 m 次后得到的 π 的近似值.

1. 令 $a_0=1, b_0=\frac{1}{\sqrt{2}}, t_0=\frac{1}{4}, p_0=1$.
2. n 从 0 到 $m-1$, 顺次计算 $a_{n+1}=\frac{a_n+b_n}{2}$, $b_{n+1}=\sqrt{a_nb_n}$, $t_{n+1}=t_n-p_n(a_n-a_{n+1})^2$, $p_{n+1}=2p_n$.
3. 返回 $\pi_m=\frac{(a_m+b_m)^2}{4t_m}$.

注92. 使用 Brent-Salamin 算法计算时, 前三次迭代可以得到近似值

$$3.14,$$
$$3.1415926,$$
$$3.14159265358979.$$

高阶迭代公式

除了利用代数几何平均值之外, 人们还发现了许多其他的高阶迭代公式, 例如下面的三个著名的迭代公式就是由 Borwein 兄弟发现的(参见 [35]).

定理5.10 (Borwein 公式). *1. 令 $x_0=\sqrt{2}$, $y_0=\sqrt[4]{2}$, $p_0=2+\sqrt{2}$. 递推公式为*

$$x_{n+1}=\frac{1}{2}(x_n^{\frac{1}{2}}+x_n^{-\frac{1}{2}}),$$
$$y_{n+1}=\frac{x_n^{\frac{1}{2}}y_n+x_n^{-\frac{1}{2}}}{y_n+1},$$
$$p_{n+1}=p_n\frac{x_{n+1}+1}{y_{n+1}+1}.$$

那么

$$|p_{n+1}-\pi|<\frac{2^{-n-1}}{\pi^2}|p_n-\pi|^2,$$
$$|p_n-\pi|<\frac{\pi^2 2^{n+4}e^{-2^{n+1}\pi}}{\mathrm{AGM}(1,\frac{1}{\sqrt{2}})^2}$$
$$<10^{-2^n}.$$

2. 令 $p_0 = 6 - 4\sqrt{2}$, $x_0 = \sqrt{2} - 1$. 递推公式为

$$\begin{aligned} x_{n+1} &= \frac{1 - (1 - x_n^4)^{\frac{1}{4}}}{1 + (1 - x_n^4)^{\frac{1}{4}}}, \\ p_{n+1} &= p_n(1 + x_{n+1})^4 - 2^{2n+3} x_{n+1}(1 + x_{n+1} + x_{n+1}^2). \end{aligned}$$

那么 $|p_n - \frac{1}{\pi}| < 4^{n+2} e^{-2^{2n+1}\pi}$.

3. 令 $a_0 = \frac{1}{3}$, $r_0 = \frac{\sqrt{3}-1}{2}$, $s_0 = (1 - r_0^3)^{\frac{1}{3}}$. 递推公式为

$$\begin{aligned} t_{n+1} &= 1 + 2r_n, \\ u_{n+1} &= (9r_n(1 + r_n + r_n^2))^{\frac{1}{3}}, \\ v_{n+1} &= t_{n+1}^2 + t_{n+1}u_{n+1} + u_{n+1}^2, \\ w_{n+1} &= \frac{27(1 + s_n + s_n^2)}{v_{n+1}}, \\ a_{n+1} &= w_{n+1}a_n + 3^{2n-1}(1 - w_{n+1}), \\ s_{n+1} &= \frac{(1 - r_n)^3}{(t_{n+1} + 2u_{n+1})v_{n+1}}, \\ r_{n+1} &= (1 - s_{n+1}^3)^{\frac{1}{3}}. \end{aligned}$$

那么 $|a_n - \frac{1}{\pi}|$*9*阶收敛.

Beeler 公式

最后值得一提的是 Beeler 利用不动点定理和反正切函数的 Taylor 展开式给出的一系列收敛到 π 的递推公式(参见 [25]).

定理5.11.

$$\begin{aligned} p_{n+1} &= p_n - \tan p_n, \\ p_{n+1} &= p_n - \tan p_n + \frac{1}{3}\tan^3 p_n, \\ p_{n+1} &= p_n - \tan p_n + \frac{1}{3}\tan^3 p_n - \frac{1}{5}\tan^5 p_n, \\ &\cdots, \end{aligned}$$

其中初始值 p_0 应该取为 π 的一个普通的近似值, 例如 $\frac{22}{7}, \frac{355}{113}$ 等.

5.2 自然对数底

5.2.1 级数方法

自然对数底常记为 e.

定义5.9 (自然对数底)**.**

$$e = \lim_{n\to\infty} e_n = \lim_{n\to\infty} (1+\frac{1}{n})^n.$$

根据 Newton 二项式公式可以得到

$$\begin{aligned} e_n = (1+\frac{1}{n})^n &= \sum_{k=0}^{n} C_n^k (\frac{1}{n})^k \\ &= \sum_{k=0}^{n} \frac{n(n-1)\cdots(n-k+1)}{k!}\cdot\frac{1}{n^k} \\ &= \sum_{k=0}^{n} \frac{1}{k!}(1-\frac{1}{n})\cdots(1-\frac{k-1}{n}). \end{aligned}$$

记

$$s_n = \sum_{k=0}^{n} \frac{1}{k!},$$

那么 $e_n < s_n \le e$, 即

$$e = \lim_{n\to\infty} s_n = \lim_{n\to\infty} 1+\frac{1}{1!}+\frac{1}{2!}+\cdots+\frac{1}{n!}.$$

进一步还可以推出

$$|e-s_n| < \frac{1}{nn!}.$$

和计算圆周率 π 用到的很多级数的误差项不同, 这个误差项很小, 它关于 n 是指数级收敛的, 因此我们可以直接利用级数 s_n 来计算自然对数底 e.

除此之外, 另一个级数

$$e^{-1} = \lim_{n\to\infty} 1-\frac{1}{1!}+\frac{1}{2!}+\cdots+(-1)^n\frac{1}{n!}$$

也可以求出 e, 它常常被用来验证第一个级数的计算结果. 这两个级数都可以使用折半求和方法来计算.

通过级数 s_n 的计算公式还可以得到关于自然对数底 e 和圆周率 π 一个有趣的递推公式.

定理5.12. 令 $u_0 = v_0 = 0$, $u_1 = v_1 = 1$, 递推公式为

$$u_{n+2} = u_{n+1} + \frac{1}{n}u_n,$$
$$v_{n+2} = \frac{1}{n}v_{n+1} + v_n,$$

那么

$$\lim_{n\to\infty} \frac{n}{u_n} = e,$$
$$\lim_{n\to\infty} \frac{2n}{v_n^2} = \pi.$$

5.3 对数常数

5.3.1 级数方法

对数常数指的是 $\ln 2$, 在计算机被发明之前, 科学家和工程师们只能借助于对数表和对数尺之类的工具来完成繁杂的乘除运算, 注意到

$$\ln x = \ln \frac{x}{2^n} + n \ln 2,$$

可以适当的选取 n 使得 $0 < \frac{x}{2^n} < 1$, 这样就需要计算 $\ln 2$ 来完成大数与标准 $(0, 1)$ 区间之间的转换.

利用 $\ln(1 + x)$ 的 Taylor 展开式

$$\ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots,$$

令 $x = 1$ 就可以得到 $\ln 2$ 的形式化定义.

定义5.10 (对数常数)**.**

$$\ln 2 = \lim_{n\to\infty} 1 - \frac{1}{2} + \frac{1}{3} + \cdots + (-1)^{n+1}\frac{1}{n}.$$

注93. 如果令 $x = -\frac{1}{2}$, 那么有

$$\ln 2 = -\ln \frac{1}{2} = \sum_{k=1}^{\infty} \frac{1}{k2^k}.$$

利用恒等式

$$\ln 2 = -\frac{1}{2}\ln(1 - \frac{1}{4}) + \tanh^{-1}\frac{1}{2},$$

还可以得到类似于计算圆周率时提到过的 BBP 公式的两个公式.

定理5.13.

$$\ln 2=\sum_{k=0}^{\infty}\frac{1}{4^k}(\frac{1}{4k+2}+\frac{1}{8k+8}),$$
$$\ln 2=\frac{2}{3}+\sum_{k=1}^{\infty}\frac{1}{16^k}(\frac{1}{16k+12}+\frac{1}{8k+4}+\frac{1}{4k+1}+\frac{1}{2k}).$$

与计算圆周率 π 类似, 计算对数常数 $\ln 2$ 也有相应的 Machin 型公式.

首先来看反正切函数的 Taylor 展开式

$$\tanh^{-1}x=\frac{1}{2}\ln\frac{1+x}{1-x}=\sum_{k=0}^{\infty}\frac{x^{2k+1}}{2k+1},$$

令 $x=\frac{1}{3}$ 就可以得到一个关于 $\ln 2$ 的 Machin 型公式.

定理5.14.

$$\ln 2=2\tanh^{-1}\frac{1}{3}=\frac{2}{3}\sum_{k=0}^{\infty}\frac{1}{(2k+1)9^k}.$$

注94. 如果考虑类似于 $\tanh^{-1}\frac{1}{x}=\tanh^{-1}\frac{1}{2x-1}+\tanh^{-1}\frac{1}{2x+1}$ 的初等变换, 还可以得到一系列 $n=2$ 的 Machin 型公式, 其中最有名的是 Euler 用来计算 $\ln 2$ 时用到的

$$\ln 2=2\tanh^{-1}\frac{1}{5}+2\tanh^{-1}\frac{1}{7}.$$

除此之外, 人们最近还发现了一些高阶的 Machin 型公式.

定理5.15.

$$\ln 2=18\coth^{-1}26-2\coth^{-1}4801+8\coth^{-1}8749,$$
$$\ln 2=144\coth^{-1}251+54\coth^{-1}449-38\coth^{-1}4801+62\coth^{-1}8749,$$
$$\ln 2=72\coth^{-1}127+54\coth^{-1}449+34\coth^{-1}4801-10\coth^{-1}8749.$$

前面介绍超几何级数时提到过

$$\begin{aligned}\tanh^{-1}x&=xF(\frac{1}{2},1,\frac{3}{2},x^2)\\&=\frac{x}{1-x^2}F(1,1,\frac{3}{2},\frac{x^2}{x^2-1}).\end{aligned}$$

定理5.16.

$$\ln 2=2\tanh^{-1}\frac{1}{3}=\frac{3}{4}(1-\frac{1}{4}\cdot\frac{1}{3}+\frac{1}{4^2}\cdot\frac{1\cdot 2}{3\cdot 5}-\frac{1}{4^3}\cdot\frac{1\cdot 2\cdot 3}{3\cdot 5\cdot 7}+\cdots).$$

注95. 这个公式结构简单, 很容易利用折半求和方法来计算它.

5.3.2 迭代方法

代数几何平均值迭代同样适用于计算对数常数 $\ln 2$(参见 [38]).

记

$$\frac{1}{R(a,b)} = 1 - \sum_{n=0}^{\infty} 2^{n-1}(a_n^2 - b_n^2),$$

可以看出 $R(a,b)$ 实际上就是 $\frac{\pi}{2\mathrm{AGM}(a,b)}$. 那么

$$|\ln x - R(1,10^{-N}) + R(1,10^{-N}x)| \le \frac{N}{10^{2(N-2)}}.$$

取 $R(1,10^{-N}) - R(1,2\cdot 10^{-N})$ 作为 $\ln 2$ 的近似值, 依然借用计算 $\mathrm{AGM}(a,b)$ 时的迭代结构, 那么在计算 $\mathrm{AGM}(a,b)$ 的同时就可以计算出 a_n, b_n, 进而可以计算出 $R(a,b)$, 这样就得到了计算 $\ln 2$ 的一个二阶收敛的迭代算法.

5.4 Euler 常数

5.4.1 级数方法

定义5.11 (Euler 常数)**.**

$$\gamma = \lim_{n\to\infty}(1 + \frac{1}{2} + \cdots + \frac{1}{n} - \ln n) =: \lim_{n\to\infty}(H_n - \ln n).$$

注96. Euler 常数常记为 γ. 如果直接按照这个定义来计算 γ 将会发现它收敛得太慢了.

利用 Euler-Maclaurin 求和对调和级数 H_n 做渐进展开可以得到关于 γ 的一个更好的近似公式.

定义5.12 (Bernoulli 数)**.** Bernoulli 数 B_n 是级数展开式

$$\frac{x}{e^x - 1} = \sum_{n=0}^{\infty} \frac{B_n x^n}{n!}$$

中第 n 项的系数.

定理5.17.

$$\gamma = H_n - \ln n - \frac{1}{2n} + \sum_{k=1}^{\infty} \frac{B_{2k}}{2k} \cdot \frac{1}{n^{2k}}, \quad B_{2k}\text{为 } Bernoulli\text{ 数},$$

它的误差项是

$$\epsilon_k, n = \frac{B_{2k+2}}{(2k+2)n^{2k+2}} \approx \frac{2(2k+2)!}{(2k+2)(2\pi n)^{2k+2}}.$$

将 $\gamma=-\Gamma'(1)$ 分部积分后可以得到

$$\gamma+\ln n=I_n-R_n,$$
$$I_n=\int_0^n\frac{1-e^{-t}}{t}\,dt,$$
$$R_n=\int_n^\infty\frac{e^{-t}}{t}\,dt.$$

注意到 $R_n=O(e^{-n})$, 利用 $\frac{1-e^{-t}}{t}$ 的级数展开式可以得到

$$I_n=\sum_{k=1}^{\infty}(-1)^{k-1}\frac{n^k}{kk!},$$

于是

$$\gamma=\sum_{k=1}^{\alpha n}(-1)^{k-1}\frac{n^k}{kk!}-\ln n+O(e^{-n}),\quad \alpha(\ln\alpha-1)=1.$$

引入常数 α 是为了使 $\frac{n^{\alpha n}}{(\alpha n)!}$ 是 e^{-n} 阶的, 其近似值为 $\alpha=3.5911$. 如果考虑 R_n 的渐进展开式, 还可以得到收敛更快的公式, 不过公式的形式同时也会变得很复杂.

使用 Bessel 函数做类似的工作, 我们将得到

$$\gamma=\frac{A_n}{B_n}-\ln n+O(e^{-4n}),$$

其中

$$A_n=\sum_{k=0}^{\alpha n}(\frac{n^k}{k!})^2H_k,$$
$$B_n=\sum_{k=0}^{\alpha n}(\frac{n^k}{k!})^2,$$
$$H_k=1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{k},$$

这里 α 的定义与上面相同, 这个级数收敛得很快, 并且很容易计算.

使用类似于 Richard 外推加速法(参见 [7])的方法对误差项进行渐进展开, 可以得到

$$\gamma=\frac{A_n}{B_n}-\frac{C_n}{B_n^2}-\ln n+O(e^{-8n}),$$

其中

$$C_n=\frac{1}{4n}\sum_{k=0}^{2n}\frac{((2k)!)^3}{(k!)^4(16n)^{2k}},$$

并且 α 满足 $\alpha(\ln\alpha-1)=3$, 其近似值为 $\alpha=4.970625759$, 利用这个公式可以计算 γ 到小数点后 1 亿位.

5.5 其他常数

下面是一些其他数学常数的定义或常用计算公式.

- Catalan 常数.

$$C = 1 - \frac{1}{3^2} + \frac{1}{5^2} - \frac{1}{7^2} + \frac{1}{9^2} - \cdots.$$

$$C = \int_0^1 \frac{\tan^{-1} x}{x} \mathrm{d}x.$$

$$C = \frac{3}{8} \sum_{k=0}^{\infty} \frac{(k!)^2}{(2k)!(2k+1)^2} + \frac{\pi \ln(2+\sqrt{3})}{8}.$$

- Brun 常数.

$$B = (\frac{1}{3} + \frac{1}{5}) + (\frac{1}{5} + \frac{1}{7}) + (\frac{1}{11} + \frac{1}{13}) + \cdots.$$

- Mertsen 常数.

$$\mu = \lim_{p \to \infty} (\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \cdots + \frac{1}{p} - \ln \ln p).$$

6 线性代数

在计算机代数系统中, 涉及线性代数的内容主要包括两方面的问题:一是线性方程组的求解, 二是矩阵各种标准形的约化. 其中线性方程组不仅作为许多应用问题的数学模型, 还是众多算法的基础, 支撑着整个通用计算机代数系统. 矩阵的标准形主要是指整环或域上矩阵的 Hermite 标准形, Smith 标准形, Frobenius 标准形以及 Jordan 标准形等[1]. 矩阵到这些标准形的约化过程与 Diophantine 方程求解以及更广泛的计算数论, 计算群论等问题有着密切的联系.

本章主要讨论快速矩阵乘法与线性方程组求解的算法. 对矩阵标准形约化算法感兴趣的读者可以参考 [80]2.3.2 节中所列出的参考文献.

6.1 快速矩阵乘法

在线性代数问题中, 矩阵乘法具有基础性的意义. 在代数复杂性理论中, 大部分线性代数问题都可约化为矩阵乘法([32]2.2 节, [48] 第 16 章), 因此矩阵乘法构成了复杂性理论中一个重要的研究模型([48] 第 15 章). 对于 n 阶一般矩阵的乘法, 由定义得到的常规算法具有 $O(n^3)$ 的复杂度[2]. 自 1968 年 Strassen[165] 发现一种基于分治策略的快速矩阵乘法算法以来, 矩阵乘法算法复杂度的指数已由 3 降到 2.376[59]. 下面我们回顾两个经典算法, 它们在实际中有着重要的应用. 而更多算法虽然渐进复杂度更低, 对于代数复杂性理论研究有着重要意义, 但由于算法过于

[1]这些矩阵标准形的定义可参考 [131].

[2]本章提到的"算法复杂度"均指乘法复杂度, 即在计算复杂度时仅计入乘法运算.采用这种计算方法是因为对于多数代数结构, 乘法运算的消耗要远远大于加减法的消耗

复杂, 且对于有限规模的问题所需运算更多, 因而并不实用, 可参考 [134] 及 [48] 第 15 章.

6.1.1 基于向量内积算法的 Winograd 算法

以下讨论主要根据文献 [186].

算法6.1 (Winograd 内积算法).

设 $x = (x_1, \cdots, x_n)^T$, $y = (y_1, \cdots, y_n)^T$, 记 $\xi = \sum_{j=1}^{\lfloor n/2 \rfloor} x_{2j-1}x_{2j}$, $\eta = \sum_{j=1}^{\lfloor n/2 \rfloor} y_{2j-1}y_{2j}$, 则内积 (x, y) 可由下式给出:

$$(x,y) = \begin{cases} \sum_{j=1}^{\lfloor n/2 \rfloor} (x_{2j-1} + y_{2j})(x_{2j} + y_{2j-1}) - \xi - \eta, & n \text{ 为偶数}, \\ \sum_{j=1}^{\lfloor n/2 \rfloor} (x_{2j-1} + y_{2j})(x_{2j} + y_{2j-1}) - \xi - \eta + x_n y_n, & n \text{ 为奇数}. \end{cases}$$

将这种算法用于 $C = AB$ 的矩阵元素运算时, 由于减少重复计算 ξ, η, 可使计算所需的乘法次数减半, 但同时使所需的加法运算增加. Winograd 算法也是 $O(n^3)$ 的算法, 仅适用于小规模的矩阵求积运算.

6.1.2 Strassen 算法

Strassen 算法是一种分治策略的算法. 它以分块矩阵运算为基础.

下面介绍改进型 Strassen 算法, 它较原始算法 [165] 需要更少的矩阵加法运算([174]12.1 节).

算法6.2 (Strassen 算法).

设 A, B 为 n 阶矩阵, 必要时通过补充零行零列加以扩充, 将其分块:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}, C = AB = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}.$$

进行如下递归运算:

1. 若 $n \le l$(l 为递归下界), 采用直接算法进行计算.

2. 计算

$$S_1 = A_{21} + A_{22}, S_2 = S_1 - A_{11}, S_3 = A_{11} - A_{21}, S_4 = A_{12} - S_2,$$
$$T_1 = B_{12} - B_{11}, T_2 = B_{22} - T_1, T_3 = B_{22} - B_{12}, T_4 = T_2 - B_{21}.$$

3. 计算

$$P_1 = A_{11}B_{11}, P_2 = A_{12}B_{21}, P_3 = S_4B_{22}, P_4 = A_{22}T_4,$$
$$P_5 = S_1T_1, P_6 = S_2T_2, P_7 = S_3T_3.$$

4. 计算

$$U_1 = P_1 + P_2, U_2 = P_1 + P_6, U_3 = U_2 + P_7, U_4 = U_2 + P_5,$$
$$U_5 = U_4 + P_3, U_6 = U_3 - P_4, U_7 = U_3 + P_5.$$

5. 返回

$$\begin{bmatrix} U_1 & U_5 \\ U_6 & U_7 \end{bmatrix}.$$

以上算法的正确性通过直接代入即可验证. 可以看出, 每次递归需要 7 次乘法与 15 次加法, 从而其算法复杂度是 $O(n^{\log_2 7}) \simeq O(n^{2.808})$.

下面考虑一个技术细节, 即对于阶数 n 不是 2^k 的矩阵添加零行零列的问题. 很容易想到两种方案, 一是在必要时才考虑添加, 即在递归过程中, 遇到矩阵阶数为奇数的情形则给它添加一个零行(或零列);二是统一添加, 即在计算的开始首先考察矩阵的阶数, 若它不满足 2^k 的要求, 即给它添加若干个零行零列, 使之满足, 而在递归过程中则不需再考虑矩阵阶数的问题. 简单的分析可以知道, 由于第一种方式将添加零行零列的工作分成许多次完成, 造成其计算效率大大下降, 因此实际中应采用第二种方法.

J. Demmel 等指出, 以 Strassen 算法为代表的快速矩阵乘法在一定意义下(所谓“弱”意义下)是数值稳定的 [67]. 我们对随机生成的浮点数矩阵进行了测试, 并与经典算法给出的结果做对比, 确实未见数值稳定性有明显下降.

可以看到, Strassen 算法与“高精度计算”中的 Karatsuba 算法的“神似”. 读者可能会提出这样的问题:能不能类似 Toom-Cook 算法一样推广 Strassen 算法到高阶从而降低算法的渐进复杂度呢?这个问题要从两方面来回答. 直接的类比和推广由于矩阵乘法不可交换而受到限制, 但将矩阵分块推广到 $n \times n$ 块重新构造快速算

法的确是可行的, 然而这种从 2 到 n 的推广却不像整数运算那么直接. 例如, 对于 $n = 3$ 的情形, Laderman[109] 给出了需要 23 次乘法的算法, 若采用递归计算, 其渐进复杂度指数为 $\log_3 23 > \log_2 7$, 反而不及 Strassen 算法. 类似的这种尝试可参考 [134], 这方面的研究及相关理论分析最终导致了双线性算法 概念及其复杂性理论的诞生. 此后对于矩阵乘法复杂度的研究主要是沿着这个方向进行, 迄今为止已经达到的最优渐进复杂度为 $O(n^{2.376})$[59]. 但在实际中, 仅当 n 极大时才有价值, 故通常并不采用. 可参考 [134][48] 及其中所引用的参考文献.

6.2 线性方程组与消元法

线性方程组求解是线性代数的中心问题之一. 它是许多实际问题的数学模型, 还构成许多数值与符号算法的基础, 从而在工程计算与计算机代数系统中都占据着重要的位置. 因此, 人们对线性方程组求解算法进行了深入的研究, 针对不同的问题提出了大量的高效算法. 其中, 数值算法主要包括一般系数矩阵的消元法以及应用于稀疏或有结构系数矩阵的算法, 读者可以参考 [76] 和 [32]. 我们在随后几节中主要介绍精确求解线性方程组的算法. 这些算法大多可直接应用于整系数, 有理系数, 多项式系数或有限域上的方程组, 并可以推广到一般整环, 商域及其扩域上.

概括说来, 精确算法可分为以下几类(其算法归纳可参考 [80]2.3 节以及 [171]), 我们将分别选取典型的算法进行介绍.

- 消元法. 这一类算法以 Gauss 消元算法为基础, 但由于直接的消元步骤对于精确计算会遭遇“中间表达式膨胀”的困难, 而往往通过特定的映射对问题进行约化.
- 黑箱算法. 这相当于数值计算中大量运用的迭代法.
- 应用于稀疏或有结构的系数矩阵的特殊算法.

关于有结构系数矩阵的线性方程组的求解, 在 [76] 和 [32] 中, 针对不同的系数矩阵(Vandermonde 阵, Toeplitz 阵和 Hilbert 阵等[3])给了详尽的讨论. 另外, 与求解线性方程组密切相关的一个问题是求解整系数不定线性方程的整数解(Diophantine 解), 它与矩阵的 Hermite 标准形约化有密切的关系, 也请读者参考 [80]2.3 节列出的有关文献.

本节将要介绍的消元法也可称为直接法, 这是相对于下节中要介绍的黑箱算法来说的. 这些消元算法以 Gauss 消元法为基础, 通过对系数矩阵元素的直接运

[3]这些特殊矩阵的定义可参考 [32].

算对矩阵进行约化, 得到方程组的解. 不同算法的区别体现在实施约化的过程. 概括来说, 本节介绍的三种算法都是通过某种映射将消元过程归结到更易计算的有限域或浮点数系统中进行. 除这些算法之外, 还有一种称为 Exact Division 算法, 是通过细致分析整系数方程组的消元过程, 从每一步消元的结果中除掉它们的公因子, 从而控制矩阵元素规模的增长, 可以参考 [24] 和 [187] 第 10 章.

6.2.1 基于中国剩余定理的消元法

我们先简要回顾求解线性方程组的一般方法, 即 Gauss 消元法, 并给出一些相关概念, 这在任何一本线性代数教材中都可以找到([12] 第 3 章). 为了讨论最一般的情形, 我们设要求解的方程组为 $AX = B$, 其中 A 为 $m \times n$ 阶矩阵, B 为 $m \times q$ 阶矩阵, 未知元排列成 $n \times q$ 的矩阵 X, 并设 A 与 B 为 s 元多项式系数的矩阵, 对于整系数矩阵只要将 s 取为 0.

根据线性代数的知识, 线性方程组的的一般解集合由一个特解与系数矩阵的零空间表征, 即该方程组的任意解都可表达为该特解与零空间中向量的和. 因此, 我们要求线性方程组的一般解, 就要同时求出其特解和零空间的一组基. Gauss 消元的基本做法是对线性方程组的系数矩阵与增广矩阵进行行初等变换, 将其化为相抵的行既约阶梯形阵(row-reduced echelon, RRE), 即如下形式(最后的 0 可能是子方阵, 也可能没有):

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

它具有如下特点:

- 非 0 行的最左非 0 元素所在列的其余元素均为 0, 且最左非 0 元素的位置随行号增加而右移;[4]
- 各非 0 行最左非 0 元素的位置, 随行号增加而右移, 若有 0 行均排在最后.

详言之, 若非零矩阵 B 满足:存在一列整数 $1 \leq k_1 < k_2 < \cdots < k_r \leq n$, 其中 $1 \leq r \leq m$(r 即矩阵 B 的秩), 使得 $B(i, k_i) \neq 0$, $B(i, j) = 0$, $1 \leq j < k_i$, 且若 $r < m$, 则第 $r+1, \ldots, m$ 行均为 0. 称 $K = \{k_1, k_2, \cdots, k_r\}$ 为 B 的既约阶

[4] 相比 [12] 中的定义, 我们放宽了每行最左非 0 元素为 1 的要求.

梯(reduced echelon, RE)序列, $B(i,k_i), 1\le i\le r$ 称为 RRE 矩阵的对角元素[5]. 若 A 与 B 行相抵, 则也称 B 为 A 的 RRE, K 为 A 的 RRE 序列. 对于已经化为这种形式的系数矩阵与增广矩阵, 我们很容易判断线性方程组是否有解并求出其一般解. 在下面我们考察的情形中, 非 0 行往往以一个公共元素 d 开始, 从而对于整系数线性方程组的求解可以避免中间计算过程出现分数.

在本小节中, 我们将介绍一种适用于整系数与多元多项式系数线性方程组求解的算法 [121], 这种算法通过同态映射将系数矩阵约化到有限域上进行消元运算. 在如下算法中, 我们需要判断给定同态映射是否可用, 其判断标准由如下定义的序列 (J_A, I_A) 表征.

为了叙述方便, 我们首先引入子阵的记号:设 $1\le i_1<i_2\cdots<i_p\le m$, $1\le j_1<j_2<\cdots<j_q\le n$. $m\times n$ 阶矩阵 $A=(a_{ij})$ 中位于第 $i_1,\cdots,i_p$ 行和第 $j_1,\cdots,j_p$ 列交叉处的元素按原序排成的方阵称为 A 的一个 $p\times q$ 阶子阵, 记为

$$A\begin{pmatrix} i_1 & \cdots & i_p\\ j_1 & \cdots & j_q\end{pmatrix}.$$

记 M_j 为矩阵 A 的前 j 列构成的子矩阵. 定义序列 $J_A=\{j_1,\cdots,j_r\}$, 其中 j_h 为最小的整数 j 使得 $\mathrm{rank}(M_j)=h$. 由于行变换不改变列向量之间的线性相关性, J 即上文定义的行 RE 序列, 而且是唯一的. 对于非零矩阵 A, 可以找到一列互异整数 $h_1,\cdots,h_r$, $1\le h_t\le m, 1\le t\le r$ 满足

$$A\begin{pmatrix} h_1 & \cdots & h_s\\ j_1 & \cdots & j_s\end{pmatrix}\neq 0, 1\le s\le r.$$

若记 $h_{r+1},\cdots,h_m$ 为其余的整数, 则 $H=(h_1,\cdots,h_m)$ 构成 $1,\cdots,m$ 的一个排列. 记所有这样的序列 H 构成集合 $\mathcal{P}_A$, 则 $\mathcal{P}_A$ 中, 存在一个按照字典序最小的序列 $I_A=\{i_1,\cdots,i_m\}$. 等价地说, 对于 $1\le t\le r$, i_t 为依次选出来的最小的整数使得这些行向量线性无关, 而对于 $r+1\le t\le m$, i_t 被按照原序排好. 若 A 为零矩阵, 我们很自然地定义 $I_A=\{1,\cdots,m\}$. 再定义 $J_A\times I_A$ 的字典序, $(J_A,I_A)>(J_B,I_B)$, 当且仅当 $J_A>J_B$, 或 $J_A=J_B, I_A>I_B$.

在定义了序列 (J_A,I_A) 之后, 我们引入如下的正则 RRE 矩阵. 对于 $m\times n$ 阶矩阵 A, 定义

$$\bar{A}_H(k,j)=\begin{cases} A\begin{pmatrix} h_1 & \cdots & & & & & h_r\\ j_1 & \cdots & j_{k-1} & j & j_{k+1} & \cdots & j_r\end{pmatrix}, & 1\le k\le r,\\ 0, & \text{其他.}\end{cases}$$

[5]这里对角元素与通常定义不同, 注意区别.

由 J_A, H_A 的定义可知, $\bar{A}_H$ 为 RRE 矩阵, 且可以证明, $\bar{A}_H$ 与 A 行相抵. 特别地, 对于 $H_A = I_A$, 将其记为 $\bar{A}$, 并称为 A 的正则 RRE(CRRE). 很显然, 这一标准形式是唯一确定的. 对于 $H = (h_1, \cdots, h_m)$, 定义

$$\delta_H(A) = A \begin{pmatrix} h_1 & \cdots & h_r \\ j_1 & \cdots & j_r \end{pmatrix},$$

若 A 为零矩阵, 则定义 $\delta_H(A) = 0$. 我们看到 $\bar{A}_H$ 的对角线元素都等于 $\delta_H(A)$. 特别地, 对于 $H = I_A$, 将 $\delta_H(A)$ 简记为 $\delta(A)$ 或 d.

该算法的整体思路, 就是计算线性方程组增广矩阵的 CRRE, 并利用其得到线性方程组的一般解. 其中, 前者是算法最核心的部分. 我们首先来看, 若已知增广矩阵的 CRRE, 怎样求线性方程组的一般解, 即一个特解和系数矩阵的零空间.

首先考察一个 RRE 矩阵 E 的零空间. 设 E 为 $m \times n$ 阶非零 RRE 矩阵, 其 RE 序列为 $J_E = \{j_1, \cdots, j_r\}, r < n$, 公共的对角元素为 d. 记 1 到 n 中除 RE 序列之外的元素为 $1 \leq k_1 < \cdots < k_{n-r}$. 如下构造 $n \times (n-r)$ 矩阵 Z,

$$Z(j_i, j) = E(i, k_j), Z(k_j, u) = \begin{cases} -d, & u = j, \\ 0, & u \neq j. \end{cases} \tag{6.1}$$

容易证明 $EZ = 0$ 且 Z 为列独立阵. 当 E 作为矩阵 A 的 RRE 形, 即存在 $m \times m$ 阶可逆阵 U 使得 $A = UE$ 时, 则 $AZ = (UE)Z = 0$, 从而 Z 给出了 A 的零空间的一组基. 利用这一结论, 可以对线性方程组的增广矩阵 $C = (A, B)$ 做出如下结论:

定理6.1. *设 $C = (A, B)$ 为线性方程组 $AX = B$ 的增广矩阵, 其中 A 为 $m \times n$ 阶矩阵, B 为 $m \times q$ 阶矩阵. 并设 C 的秩为 r, E 为 C 的 RRE 形, 公共的对角元为 d. 当线性方程组有解时, 设 Z' 为根据 E 与 J_C 按如上方式构造出的矩阵, 设 Z 为 Z' 的前 n 行与前 $n-r$ 列的子阵, Y 为 Z' 的前 n 行与后 q 列的子阵, 则 (d, Y, Z) 是 $AX = B$ 的一般解, 即 $AY = dB$ 且 Z 张成 A 的零空间.*

证明. 依定理所述, 将 Z' 划分为 $\begin{bmatrix} Z & Y \\ U & V \end{bmatrix}$, 则应用前述结论可得

$$CZ' = (AZ + BU, AY + BV) = (AZ, AY - dB) = 0.$$

Z 的列无关性容易验证. □

然而, 对于整系数线性方程组, 如果直接对整系数矩阵进行行变换, 并保持中间表达式为整系数, 则很容易出现类似“高精度运算”部分中提到过的中间表达式膨胀(intermediate expression swell)的问题, 即经过多次乘法后, 中间表达式的位

数急速增长, 从而严重影响程序执行的效率. 多项式系数的矩阵也有类似情形. 为了避免此问题, 自 20 世纪 60 年代起已经发展了系统的模算法, 即通过如下两种同态映射:$\mathbb{Z}$ 到 F_p 的模同态与 $F_p[x_1,\dots,x_s]$ 到 $F_p[x_1,\dots,x_{s-1}]$ 的计值运算(这也等价于 $F_p[x_1,\dots,x_s]$ 模 x_s-a_s 的同态映射), 将整数环与多项式环上的问题化到有限域 F_p 上, 然后在 F_p 上执行 Gauss 消元步骤, 这样就限制了主要计算的规模. 由于矩阵的初等变换归根到底只涉及矩阵元素的乘法与加减法[6], 因此模映射使得如下的图交换:

$$\begin{array}{ccccc} M_n(\mathbb{Z}[x_1,\dots,x_s]) & \xrightarrow{\bmod p} & M_n(F_p[x_1,\dots,x_s]) & \xrightarrow{\text{计值}} & M_n(F_p) \\ \downarrow \text{行变换} & & \downarrow \text{行变换} & & \downarrow \text{行变换} \\ M_n(\mathbb{Z}[x_1,\dots,x_s]) & \xrightarrow{\bmod p} & M_n(F_p[x_1,\dots,x_s]) & \xrightarrow{\text{计值}} & M_n(F_p) \end{array} \tag{6.2}$$

为了从以上计算的结果得到有理数解或有理分式解, 可采用模同态的中国剩余定理与多项式插值来“恢复”应有结果. 这种思路可以判断线性方程组是否有解并求得其一般解. 整体来讲, 我们的算法包含如下的三个层次, 相应于 (6.2) 中的三个映射:

1. (算法的最外层)利用模映射, 将 $\mathbb{Z}$ 上的多元多项式系数的矩阵映射到 F_p 上的多元多项式系数的矩阵, 引用中层算法得到其标准行阶梯形, 再通过中国剩余定理得到整系数或有理系数多元多项式矩阵的 RRE. 据此得到线性方程组的通解;

2. (算法的中层, 应用于多元多项式系数的线性方程组)对于 F_p 上的多元多项式系数的矩阵, 通过计值映射将其化为 F_p 上的矩阵, 引用内层算法将其化为 RRE, 再通过多元多项式的插值算法得到 F_p 上的多元多项式系数的 RRE 矩阵;

3. (算法的内层)通过 F_p 上的 Gauss 消去法, 将 F_p 上的矩阵化为 RRE.

[6]以下的算法步骤中并不显式包含整数的除法, 这是我们构造该交换图的基础.然而, 必须注意, 这里出现的三个”行变换”是执行完全相同的操作, 在这个意义下, 该交换图总是成立的.而后面将会讨论的模同态的交换性是在如下意义下:对于一个已知的矩阵, 要通过行变换求得其 REE 形, 所需的变换对于模映射前后的两个矩阵可能是不同的, 例如矩阵

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

在模 p 映射下得到零矩阵, 二者的 RE 序列显然不同, 从而化为 REE 形所需行变换也是不同的.如果模映射前后的两个矩阵需要执行的行变换完全相同, 则模映射自然是交换的, 这构成”交换”的充分条件.对此, 后面会给出更为详细的分析.

对这个思路可以打一个比方:为了减小“信息处理”的工作量, 我们先对原问题中需要处理的 $\mathbb{Z}$ 上的多元多项式矩阵先做一个“压缩”映射, 对这个“压缩包”进行相应的处理之后, 再通过“解压缩”将其恢复为我们需要的处理之后的结果. 这时, 我们将面临一个重要问题, 即怎样保证这样的压缩处理是“无损压缩”, 从而可以从压缩包中重新恢复我们需要的结果呢?也就是说, 我们使用怎样的模映射, 才能保证可以通过中国剩余定理与插值算法得到原矩阵的 RRE 呢?详细说来, 可以归纳为如下的三个问题:

1. 我们所谓“无损压缩”, 究竟保持什么“信息”稳定, 从而可以据此恢复应有结果?

2. 什么样的模映射构成我们所需的“无损压缩”?

3. 我们需要多少这样的模映射才能得到真正的“无损压缩”?实际存在的这种模映射够不够用?特别地, 对于 $F_p[X]$ 到 F_p 的约化, 这个考虑是很重要的.

下面我们回答这些问题, 并给出线性方程组的求解详细算法.

首先讨论脚注 3 中提到的模映射的交换性. 引入如下记号:以 $\theta : A \longmapsto A^*$ 表示模映射, 以 $\Gamma : A \longmapsto \bar{A}$ 表示计算 REE 形的行变换. 若 θ 与 Γ 可交换, 则称模映射 θ 具有交换性. 在这种情况下, 我们对 A^* 进行行变换得到 $\bar{A}^* = (\bar{A})^*$, 从而可以对 $\bar{A}^*$ 应用中国剩余定理恢复原矩阵 A 的 REE 形. 下述定理给出了模映射的交换性的充分条件:

定理6.2. *设 A 为 $m \times n$ 阶矩阵, 在模映射 θ 下的像为 $A^* = \theta(A)$. 若 $(J_A, I_A) = (J_{A^*}, I_{A^*})$, 则 $\bar{A}^* = (\bar{A})^*$, 即 θ 可交换.*

证明. 注意到在模映射下, 求子矩阵的行列式与模映射可交换. 利用 J_A, I_A 的定义可知定理成立. □

我们称满足 $(J_A, I_A) = (J_{A^*}, I_{A^*})$ 的模映射称为“可行的(accepted)”, 否则称为不可行的(rejected). 注意到可行性并非模映射交换的必要条件, 例如下述矩阵

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

在模 p 映射下不满足上述条件, 但模 p 映射对于 A 交换. 尽管如此, 我们仍然得到了一个有效的判别条件.

下面的论述表明, 不可行的模映射是有限的. 在模映射下, 矩阵子阵的行列式只可能从非 0 映为 0. 根据矩阵秩以及 J_A, I_A 的定义可知, 必有 $r(A^*) < r(A)$ 或 $(J_{A^*}, I_{A^*}) \geq (J_A, I_A)$. 模映射 θ 可行当且仅当对于 $1 \leq k \leq r(A)$,

$$A^* \begin{pmatrix} i_1 & \cdots & i_k \\ j_1 & \cdots & j_k \end{pmatrix} \neq 0.$$

换句话说, 模映射 θ 不可行当且仅当对某个 k,

$$A^* \begin{pmatrix} i_1 & \cdots & i_k \\ j_1 & \cdots & j_k \end{pmatrix} = 0,$$

这样的模映射在整数环与多项式环上都是有限的, 分别由整数的素因子个数与多项式的次数给出上限. 因此, 我们可以期望能够得到充分多的可行的模映射完成计算. 当然, 这里的"充分多"由中国剩余定理与多项式插值公式来决定.

然而, 以上给出的条件依赖于事先知道 A 的 (J_A, I_A), 而这是不可能的. 我们在计算中已知的只能是模映射后的矩阵的 RE 序列等. 因此, 我们还需要将以上条件换成其他等价条件, 用模映射后的矩阵表达出来. 确切地说, 有以下两个定理, 它们分别处理了对 J_A 和 I_A 的检测.

定理6.3. *设 C 为 $m \times n$ 阶非零矩阵, 秩 $r < n$. E 为 $m \times n$ 阶 RRE 矩阵, 其秩 $r' < r$ 或 $r' = r, J_E \geq J_C$. 设 Z 为由 E 根据 (6.1) 构造出来的矩阵. 若 $CZ = 0$, 则 $J_C = J_E$.*

证明. 若 $r' < r$ 或 $r' = r, J_E > J_C$, 则很容易通过代入得到不可能有 $CZ = 0$ 的结果. □

根据这一定理, 我们在计算中, 只要将每步得到的 Z 代入 $CZ = 0$ 进行验证即可. 通过如下的构造, 可以减少该验证步骤的计算. 设 RRE 形矩阵 E 的 RE 序列为 $J_E = (j_1, \cdots, j_r)$, 其余行记为 $k_1, \cdots, k_{n-r}$, 公共对角元为 d, 定义 E 的非对角部分 $\tilde{E}$ 如下:

$$\tilde{E} = E \begin{pmatrix} 1 & \cdots & r \\ k_1 & \cdots & k_{n-r} \end{pmatrix}, \tag{6.3}$$

我们知道, E 中其余部分除对角元 d 外均为 0 元素. 又定义 C 的两个子阵如下:

$$C' = \begin{pmatrix} 1 & \cdots & m \\ j_1 & \cdots & j_r \end{pmatrix}, \quad C'' = \begin{pmatrix} 1 & \cdots & m \\ k_1 & \cdots & k_{n-r} \end{pmatrix}.$$

若 Z 定义如上, 则 $CZ = C'\tilde{E} - dC''$, 从而与代入 $CZ = 0$ 验证等价的条件是

$$C\tilde{E} = dC''. \tag{6.4}$$

我们将采用此式进行代入验证.

定理6.4. 设 C 为 $F_p[x_1,\cdots,x_s], s>1$ 上的矩阵, $b+1$ 为根据插值公式的次数要求给出的所需计值点数的上界(将在下面给出). 设 Ψ_{a_i} 为点 a_i 处的计值映射, $0\leq i\leq b$, 且 $C^{(i)}=\Psi_{a_i}(C)$ 的 RE 序列 $J_{C^{(i)}}=J_C$, 而 $I_{C^{(i)}}=H$. 则 $H=I_C$. 进一步, 若 E 是根据 $C^{\overline{(0)}},\cdots,C^{\overline{(b)}}$ 运用插值方法构造出的 RRE 矩阵, 则 $E=\bar{C}$.

证明. 反证法. 记 $H=(h_1,\cdots,h_m), I_C=(i_1,\cdots,i_m)$. 若 $H\neq I_C$, 不妨设最先有 $h_k\neq i_k$, 也即 $h_k>i_k$, 显然应有 $1\leq k\leq r$. 则存在 $b+1$ 个计值映射 Ψ_{a_i} 使得

$$\Psi_{a_i}(\delta_k(C))=\Psi_{a_i}\left(\det C\begin{pmatrix}i_1 & \cdots & i_k\\ j_1 & \cdots & j_k\end{pmatrix}\right)=0.$$

但由于不可行映射数目的限制(也即上述方程解个数的限制), 由 $\deg(\delta_k(C))\leq b$ 知道这是不可能的. 从而 $H=I_C$. 根据映射的交换性即得由插值方法构造的 $E=\bar{C}$. □

上述定理给出了算法过程中的另一个检验步骤.

设 $C=(A,B)$ 为 $m\times n+q$ 阶增广矩阵, 且 $\tilde{E}$ 与 d 满足 (6.4) . 则方程组 $AX=B$ 的一般解 (d,Y,Z) 可以如下构造:

$$Z(j_i,j)=\tilde{E}(i,j),\quad Z(k_j,u)=\begin{cases}-d, & u=j,\\ 0, & u\neq j;\end{cases}\tag{6.5}$$

$$Y(j_i,j)=\tilde{E}(i,n-r+j),\quad Y(k_j,u)=0.\tag{6.6}$$

下面我们给出插值方法所需的模同态的个数, 也即给出矩阵中某些子行列式或多项式次数的上界. 以下命题的证明都不困难, 只要注意到细节问题即可, 故略去. 读者可以参考 [121].

定理6.5. 设 B 为 $I[x]$ 上的 $m\times m$ 阶矩阵, 则对于任意整数 $i:1\leq i\leq m$,

$$\deg(\det(B))\leq\max_{1\leq k\leq m}\deg(B(i,k))+\sum_{u=1}^{m}e_u-\min_{1\leq u\leq m}e_u,$$

其中, $e_u=\max\limits_{1\leq t\leq m,t\neq i}\deg(B(t,u)), 1\leq u\leq m$.

定理6.6. 设 C 为 $m\times n$ 阶非零矩阵, $J_C=(j_1,\cdots,j_r)$. 定义 $f=\max\limits_{j\neq j_t}\max\limits_{1\leq i\leq m}\deg(C(i,j))$, $e_u=\max\limits_{1\leq i\leq m}\deg(C(i,j_u)), 1\leq u\leq r$, 以及 $e\sum\limits_{u=1}^{r}e_u, e_0=\min e_u$, 令 $g=f-e_0$. 定义

$$b=\begin{cases}e, & g\leq 0,\\ e+g, & g>0.\end{cases}$$

则对任意 $H\in\mathcal{P}_C$, b 构成 $\bar{C}_H$ 中元素次数的上界.

以上两个定理提供了插值算法所需的对于多项式矩阵的元素次数的估计. 一般来讲, 中国剩余定理也要求相应的元素上界估计[7]. 然而, 在本算法中由于有了 (6.4) 的代入检验, 就不再需要上界估计了.

这样, 我们可将算法整理叙述如下.

算法6.3 (最外层算法 PLES).

输入:$\mathbb{Z}[x_1,\cdots,x_s], s\le 0$ 上的 $m\times n'$ 阶矩阵 C, 整数 $n: 1\le n<n'$. 其中, $C=(A,B)$ 为线性方程组 $AX=B$ 的增广矩阵, A 为 $m\times n$ 阶非零矩阵, B 为 $m\times q$ 阶矩阵, $q=n'-n$. 我们还需要一个相当规模的素数库 $\mathcal{L}$.

输出:三元组 (d,Y,Z). 若线性方程组有解, 则 (d,Y) 构成 $AX=B$ 的特解, Z 为 A 的零空间的基. 若线性方程组无解, 则 d,Y,Z 全部设为 0.

1. (初始化)置 $r=0$.
2. (模 p 映射)若前述计算已经穷尽素数库 $\mathcal{L}$, 则算法失败. 否则, 取出下一个素数 $p\in\mathcal{L}$, 计算 $C^*\equiv C \pmod p$. 若 C^* 为零矩阵, 转到第 2 步.
3. (应用计值插值算法)运用 CPRRE 算法, 计算 $d^*=\delta(C^*), J^*=J_{C^*}, I^*=I_{C^*}$ 以及 $\bar{C}^*$ 的非对角部分 W^*.
4. (进行模算法的可行性检测)置 $r^*=\mathrm{rank}(C^*)$. 若 $J^*(r^*)>n$, 置 $d=0,Y=0,Z=0$, 返回 (d,Y,Z), 此时方程组无解. 若 $r^*>r$, 转至第 5 步. 若 $r^*<r$, 转至第 2 步. 若 $(J^*,I^*)<(J,I)$, 转至第 5 步;若 $(J^*,I^*)>(J,I)$, 转至第 2 步. 其余情形, 转至第 6 步.
5. (中国剩余定理算法初始化)置 $r=r^*,d=0,J=J^*,I=I^*,h=1$, 置 W 为 $r\times(n'-r)$ 阶零矩阵.
6. 由中国剩余定理的算法迭代步骤, 由 d,d^*,h,p 计算 d'. 采用类似的步骤, 由 W,W^*,h,p 计算 $r\times(n'-r)$ 阶矩阵 W' 的元素. 置 h 为 $p\cdot h$.
7. (相等检验)若 $d'=d$, $W'=W$, 转至第 8 步. 否则, 置 $d=d'$, $W=W'$, 转至第 2 步.
8. (代入检验)根据 (6.3) , 由 C 与 J 构造 C' 与 C''. 计算 $C'W$ 与 $d\cdot C''$. 若 $C'W\ne d\cdot C''$, 转至第 2 步.

[7]这种估计往往由 Hadamard 不等式提供, 见"Hensel 提升方法"一节.

9. (构造一般解)若 $r<n$, 根据 (6.5) , 由 J,d,W 构造 $r\times(n-r)$ 阶矩阵 Z;若 $r=n$, 置 Z 为零矩阵. 根据 (6.6) 构造 $n\times q$ 阶矩阵 Y. 返回 (d,Y,Z).

注97. 通常素数库 $\mathcal{L}$ 中的素数 p 满足 $p^2\lesssim\ell$, 其中 ℓ 为计算机硬件计算的字长, 从而保证计算中能够充分利用机器精度运算的能力. 这样的素数库可以预先存储在程序中, 也可以动态生成. 该库的生成并非本算法讨论的内容.

注98. 前述"相等检验"通过是中国剩余定理算法结束的必要条件(而不是充分条件). 此处引入该检验, 是为了让随后的"代入检验"成功的可能性更大, 从而减少代入检验的运算量.

下面的算法给出 $F_p[x_1,\cdots,x_s],s\geq 0$ 上矩阵 RRE 形的计算, 其中大部分步骤与 PLES 算法是类似的.

算法6.4 (中层算法 CPRRE).

输入:$F_p[x_1,\cdots,x_s],s\geq 0$ 上的 $n\times n$ 阶非零矩阵 C.

输出:$d=\delta(C),J=J_C,I=I_C$, 以及 $\bar{C}$ 的非对角部分 W.

1. (应用 Gauss 消去法)若 C 为 F_p 上的矩阵, 由下面介绍的 CRRE 算法计算 $d=\delta(C),J=J_C,I=I_C$ 与 W.
2. (算法初始化)置 $r=0,a=p,k=0$.
3. (计值同态)置 $a=a-1$. 若 $a<0$, 则算法失败返回. 否则, 置 $C^*=\Psi_a(C)$. 若 $C^*=0$, 转至第 3 步.
4. 递归调用 CPRRE 算法, 计算关于 C^* 的相应结果:$d^*=\delta(C^*),J^*=J_{C^*},I^*=I_{C^*}$ 以及 $\bar{C}^*$ 的非对角部分 W^*.
5. (计值同态的可行性检测)置 $r^*=\mathrm{rank}(C^*)$. 若 $r^*>r$, 转至第 6 步;若 $r^*<r$, 转至第 3 步. 若 $(J^*,I^*)<(J,I)$, 转至第 6 步;若 $(J^*,I^*)>(J,I)$, 转至第 3 步. 其余情况, 转至第 7 步.
6. (插值算法初始化)置 $r=r^*,d=0,J=J^*,I=I^*$. 若 $r<n$, 构造 $r\times(n-r)$ 阶零矩阵 W, 若 $r=n$, 置 $W=0$. 置 $E(x)=1$.
7. 通过插值算法的迭代步骤, 由 $d,d^*,E(x),a$ 构造 d'. 若 $r<n$, 通过类似的计算用 $W,W^*,E(x),a$ 构造 W'. 若 $r=n$, 置 $W'=0$. 更新 $E(x)\leftarrow E(x)\cdot(x-a)$. 若 $k=1$, 转至第 11 步.

8. (相等检验)若 $d'=d$, $W'=W$, 转至第 9 步. 否则, 置 $d=d',W=W'$, 转至第 3 步.

9. (代入检验)若 $r=n$, 转至第 10 步. 根据 (6.3) , 由 C 和 J 构造 C' 和 C''. 代入 $C'W$ 与 $d\cdot C''$ 中进行检验, 若 $C'W\neq d\cdot C''$, 转至第 3 步, 否则置 $k=1$.

10. (次数上界)根据 6.6 计算 DRRE 形矩阵元素次数的上界 b.

11. (次数检验)若 $\deg(E(x))\leq b$, 转到第 3 步. 否则, 置 $d=d',W=W'$, 返回.

注99. 注意到在算法的第 3 步中可能有算法失败返回的结果. 这里包含一个与 PLES 算法很重要的不同:由于 PLES 算法中被模掉的素数可以在全体整数中选取, 因此其算法主体总能成功(只要它调用的 CPREE 算法成功);然而, 由于 CPREE 算法的计值点只能在 F_p 中选取, 而这是有限的, 因此可能不成功. 鉴于此, 通常我们选取模同态时, 要求模掉的素数 p 充分大:同时满足 $p^2\lesssim\ell$ 的条件.

在算法的最内层, 我们应用 F_p 上的 Gauss 消去法给出矩阵的 REE 形. 由于这是一个标准的算法, 我们只要给出几个关键步骤的详细描述即可.

算法6.5 (内层算法, CRRE).

输入:F_p 上的 $m\times n$ 阶矩阵 C.

输出:表征 C 的 RRE 形的 $\delta(C),J_C,I_C$, 以及非对角元素 W.

1. (初始化)置 $J=\varnothing,I=(1,\cdots,m),d=1$. 记 $C^{(0)}=C$.

2. (向前消去步骤)设在向前消去的前 $k-1$ 步, 我们已经得到了 $C^{(k-1)}$, 并各有 $k-1$ 个元素添加到 J 与 I 中. 在向前消去的第 k 步, 执行如下步骤:

   - (寻找主元)在第 $k,k+1,\cdots,m$ 行中, 按自左向右逐列扫描的顺序, 找到第一个非零元素 $C^{(k-1)}(t,s)$. 将 s 添加到 J 中, 即令 $j_k=J_C(k)=s$;置 $d\leftarrow d\cdot C^{(k-1)}(t,s)$;交换第 k 行至第 m 行的排列顺序得到 $C^{(k)}$ 的一个"预备"形式:将第 t 行排到第 k 行, 其余各行依原序排列, 也即进行如下置换:
   $$\tau_k(h)=\begin{cases}h, & 1\leq h<k\text{或}t<h\leq m,\\ k, & h=t,\\ h+1, & k\leq h<t.\end{cases}$$
   随后将 I 中的元素进行类似的重排:将 $I(\tau_k(h))$ 换成 $I(h)$.

- 令 $e=(C^{(k)}(t,s))^{-1}$, $C^{(k)}(k,j)\leftarrow C^{(k)}(k,j)\cdot e, s\le j\le n$.
- (向前消去)对于 $h>k$ 行中第 s 列元素非零者, $C^{(k)}(h,j)\leftarrow C^{(k)}(h,j)-C^{(k)}(h,s)\cdot C^{(k)}(k,j), s\le j\le n$.

3. (向后消去)经过向前消去后, 我们已经得到了 C 的阶梯形式. 设 $\mathrm{rank}(C)=r$, 则须执行 $r-1$ 个向后消去步骤以得到 C 的 RRE 形式. 在向后消去的第 $k:1\le k\le r-1$ 步, 利用第 $r-k+1$ 行将前 $r-k$ 行的第 j_{r-k+1} 列元素消去. 可以证明, 由此得到的 RRE 矩阵记为 $\bar{C}$, 据此可以得到其非对角元素 W.

至此, 我们已完成了全部算法的叙述. McClellan 在 [121] 中给出了详细的算法复杂性分析, 我们不再给出具体结果. 我们仅指出, 这种算法的实际执行效率高度依赖于素数库的选择与插值映射的选择. 对于 $m\times n$ 阶系数矩阵的线性方程组, 其平均时间效率与借助整数严格除法执行的 Exact Division 算法等相比, 大致提高了 m 倍.

6.2.2 Padé 逼近与有理函数重建

由于下面将要介绍的几个算法都要通过一定形式的“逼近”与“恢复”过程, 它们都以 Padé 逼近以及有理函数重建算法为基础. 我们先来介绍这两个问题.

有理函数重建

有理函数重建(Rational Function Reconstruction)要解决的问题是:对于域上 n 次多项式 $m\in F[x]$ 以及次数小于 n 的多项式 $f\in F[x]$, 我们要找到多项式 $r,t\in F[x]$ 使得有理函数 $r/t\in F(x)$ 满足 $\gcd(t,m)=1$ 且

$$rt^{-1}\equiv f \bmod m,\quad \deg r<k,\quad \deg t\le n-k, \tag{6.7}$$

其中 t^{-1} 是指 t 在模 m 下的逆. 如果将 k 取为 n, 则 $r=f$, $t=1$ 显然是该问题的一个解. 如果我们不考虑 $\gcd(t,m)=1$ 的约束, 则问题可以等效需要满足下面一个较弱的条件

$$r\equiv tf \bmod m,\quad \deg r<k,\quad \deg t\le n-k. \tag{6.8}$$

下面给出一个引理.

引理6.1. 设有多项式 $f,g,r,s,t\in F[x]$ 满足 $\deg f=n$, $r=sf+tg$, $t\neq 0$ 并且

$$\deg r+\deg t<n=\deg f.$$

再令 $r_i, s_i, t_i (0 \le i \le l+1)$ 是扩展 *Euclid* 算法中相应的多项式*(*各多项式定义见 *8.3.2* 节开头*)*, 设 $j \le \{1, \dots, l+1\}$ 满足 $\deg r_j \le \deg r < \deg r_{j-1}$, 则存在非零多项式 $\alpha \in F[x]$ 满足

$$r = \alpha r_j, \quad s = \alpha s_j, \quad t = \alpha t_j.$$

证明. 首先必然成立 $st_j = ts_j$, 若不然, 则有关于 f, g 非奇异的线性方程

$$\begin{pmatrix} s_j & t_j \\ s & t \end{pmatrix} \begin{pmatrix} f \\ g \end{pmatrix} = \begin{pmatrix} r_j \\ r \end{pmatrix},$$

于是 $f = \dfrac{r_j t - r t_j}{s_j t - s t_j}$, 而

$$\begin{aligned} \deg \frac{r_j t - r t_j}{s_j t - s t_j} &\le \deg(r_j t - r t_j) \le \max\{\deg r_j + \deg t, \deg r + \deg t_j\} \\ &\le \max\{\deg r + \deg t, \deg r + n - \deg r_{j-1}\} \\ &< \max\{n, \deg r_{j-1} + n - \deg r_{j-1}\} = n = \deg f, \end{aligned}$$

注意其中用到了 $\deg t_j = n - \deg r_{j-1}$, 此式可以用辗转相除的过程归纳证明. 于是, 我们得到了矛盾 $\deg f < \deg f$, 因而 $s_j t = s t_j$.

由 [174] 引理 3.8 知 $\gcd(s_j, t_j) = 1$, 由此可知 $t_j \mid s_j t \Rightarrow t_j \mid t$, 于是非零多项式 $\alpha \in F[x]$ 使得 $t = \alpha t_j$, 由 $st_j = s_j t = \alpha s_j t_j$ 得到 $s = \alpha s_j$, 从而

$$r = sf + tg = \alpha(s_j f + t_j g) = \alpha r_j.$$

证毕. □

定义6.1 (有理函数正则形式)**.** 有理函数 $r/t \in F(x)$ 称为正则的, 如果 $r, t \in F[x]$ 满足 t 首一且 $\gcd(r, t) = 1$.

下面的定理给出了扩展 Euclid 算法和有理函数重建问题的关系, 同时也给出了有理函数重建算法.

定理6.7. *设有 n 次多项式 $m \in F[x]$ 和次数小于 n 的多项式 $f \in F[x]$, r_j, s_j, t_j 的定义同引理 6.1 , 这里取为关于 f 和 m 的扩展 Euclid 算法中的各个多项式(相当于令 $g = m$), 其中 j 取为最小的整数使得 $\deg r_j < k$, 那么:*

1. *存在多项式 r, t 满足 (6.8) 式, 即 $r = r_j$, $t = t_j$, 若还有 $\gcd(r_j, t_j) = 1$, 则 r, t 也满足 (6.8) 式, 即求得了有理函数重建问题的解.*

2. *若 $r_j/t_j \in F(x)$ 满足 (6.8) , 设 $\tau = \mathrm{lc}(t_j) \neq 0$, 令 $r = r_j/\tau$, $t = t_j/\tau$, 那么 r/t 是一组正则解. 特别地, 有理函数重建问题可解当且仅当 $\gcd(r_j, t_j) = 1$.*

证明过程较容易, 用到了扩展 Euclid 算法内容以及引理 6.1 . 有兴趣的读者可以参考 [174]5.7 节.

Padé 逼近

已知域上一形式幂级数 $f \in F[[x]]$, 那么 Padé 逼近(Padé Approximation)要解决的问题是求一有理函数 $\rho = r/t \in F(x)$ 去逼近该级数, 即 ρ 的 Taylor 展开中能有足够多的 x 的幂次与 f 符合.

定义6.2 (Padé 逼近). 设 $f \in F[[x]]$, 如果有理函数 r/t 满足

$$x \nmid t, \quad r/t \equiv f \bmod x^n, \quad \deg r < k, \quad \deg t \leq n-k, \tag{6.9}$$

则称 r/t 是 f 的一个 $(k, n-k)$-Padé 逼近.

注意到如果令 $m = x^n$, 则 Padé 逼近问题 (6.9) 化为有理函数重建问题 (6.7) . 因此, 我们完全采用上一节的算法就可以求出 Padé 逼近了. 关于 Padé 逼近性质及其算法更详细的介绍可以参考 [5].

6.2.3 Hensel 提升算法

求解非奇异整系数线性方程组 $Ax = b$ 的模算法的另一种思路是对解进行 p 进制(p-adic, p 为素数)展开, 然后“恢复”精确的有理数解. 这一算法相当于多项式计算中介绍的 Hensel 提升算法, 是 Moenck, Carter[123] 与 Dixon[69] 提出的. 本小节介绍 Dixon 提出的应用于整系数方程组的算法, 又称为 p-adic 算法. 对于一元多项式系数的方程组, 可完全类似地利用 Hensel 提升实施约化过程, 再借助一元多项式的 Padé 逼近得到有理函数解, 这就是一般的 Moenck-Carter 算法.

该算法包括三个主要步骤:

1. 合理选定素数 p, 要求 $p \nmid \det A$. 在 F_p 上计算 A 的逆 C, 即 $AC \equiv I \pmod p$, C 的存在性由 $C \equiv A^*/\det A \pmod p$ 保证, 其中 A^* 是 A 的古典伴随方阵. 这一步可以利用 F_p 上的 Gauss 消元法实现, 也可采用其他有限域上求逆的算法得到.

2. 对于选定的充分大的 m, 计算 $\bar{x}$, 使得 $A\bar{x} \equiv b \pmod{p^m}$. $\bar{x}$ 称为 x 的 p 进展式.

3. 利用连分式展开的方法(如下所述)由 x 的 p 进展式得到其有理解.

首先考察如何得到 x 的 p 进展式. 执行如下步骤:

$$\begin{aligned} b_0 &\leftarrow b, \\ x_i &\leftarrow Cb_i \pmod{p}, \\ b_{i+1} &\leftarrow p^{-1}(b_i - Ax_i), i = 0, 1, 2, \ldots. \end{aligned}$$

注意到由于 $b_i - Ax_i = b_i - ACb_i \equiv 0 \pmod{p}$, 故上式最后一步中的除法是严格的, 所得到的 b_{i+1} 为整数. 这样的循环步骤将在计算得到 x_{m-1} 与 b_m 之后结束(m 要取多大在后面给出). 这时, 令

$$\bar{x} = \sum_{i=0}^{m-1} x_i p^i,$$

我们有

$$A\bar{x} = \sum_{i=0}^{m-1} p^i A x_i = \sum_{i=0}^{m-1} p^i (b_i - p b_{i+1}) = b_0 - p^m b_m,$$

从而得到

$$A\bar{x} \equiv b \pmod{p^m}.$$

为了从 $\bar{x}$ 得到 x, 注意到 x 一般为有理系数向量, 可以表达为 $x = f/g$, 其中 f 为整系数向量, $g|\det A$. 由此可得 $g\bar{x} \equiv f \pmod{p^m}$. 有如下定理:

定理6.8. *设 $s, h > 1$ 为整数, 存在整数 f, g 满足*

$$gs \equiv f \pmod{h}, \text{且} |f|, |g| \leq \lambda h^{\frac{1}{2}},$$

其中 $\lambda = 0.618\ldots$ 为方程 $\lambda^2 + \lambda - 1 = 0$ 的一个根. 设既约分数 $w_i/v_i (i = 1, 2, \ldots)$ 为 s/h 的连分式展式序列, 并记 $u_i = v_i s - w_i h$. 若该序列中 k 第一个满足 $|u_k| < h^{\frac{1}{2}}$, 则 $f/g = u_k/v_k$.

证明. 根据连分式展式的性质, 我们知道序列 $\{w_i\}$ 与 $\{v_i\}$ 递增而 $\{u_i\}$ 则正负交替而绝对值递降, w_i/v_i 收敛于 s/h. 关于连分式展开的基本性质, 可以参考 [150].

记 $f = gs - th$, 则

$$\left| \frac{s}{h} - \frac{t}{g} = \frac{fg}{hg^2} < \frac{1}{2g^2}, \right|$$

从而对任意 $t', g' < g$, 由

$$\begin{aligned}\left|\frac{s}{h}-\frac{t'}{g'}\right| &= \left|\frac{s}{h}-\frac{t}{g}+\frac{t}{g}-\frac{t'}{g'}\right| \\ &\geq \left|\frac{t}{g}-t'g'\right| - \left|\frac{s}{h}-\frac{t}{g}\right| \\ &\geq \frac{1}{gg'}-\frac{1}{2g^2} \\ &\geq \frac{1}{2g^2}\end{aligned}$$

得到, t/g 为 s/h 的一个最佳逼近, 这只能是 t/g 等于 s/h 的一个连分式展式([150] 第二章定理 1), 记为 w_j/v_j. 由于 w_j 与 v_j 互素, $|u_j| \leq |f| \leq \lambda h^{\frac{1}{2}}$, 由 k 的定义知 $j \geq k$. 另一方面, 由 $u_j = v_j s - w_j h$ 及 $u_k = v_k s - w_k h$ 得到 $u_j v_k - u_k v_j \equiv 0 \pmod{h}$. 由于 $j \geq k$, $|uj v_k - u_k v_j| \leq (|u_j| + |u_k|)|v_j| < \lambda(\lambda+1)h = h$, 从而 $u_j v_k = u_k v_j$, 即 $j = k$. 从而 $f/g = u_k/v_k$. 证毕. □

在上述定理中取 $h = p^m$, 即可给出求 f/g 的一种算法. 其中 m 由定理中要求的不等式定义:$\delta \leq \lambda p^{m/2}$, 其中 δ 为 g 与 f 的元素绝对值的上界, 可由 Hadamard 不等式(定理 10.4)给出:对任意实系数 n 阶可逆阵 $B = (b_{ij})$ 有

$$|\det B| \leq \prod_{i=1}^{n}\left(\sum_{k=1}^{n} b_{ki}^2\right)^{1/2} = \prod_{i=1}^{n} l_i,$$

其中 l_i 为列向量的 2-范数, 即 Eulid 长度. 由 Cramer 法则知道, g 与 f 中的元素均为增广矩阵的 n 阶子式, 从而可以选取 $n+1$ 个列向量中最长的 n 个向量的长度求得 Hadamard 界δ, 并计算得到 $m = \lceil 2\log(\delta\lambda^{-1})/\log p \rceil$. 随后通过连分式展开的步骤, s 取为 $\bar{x}$ 的元素, 执行如下过程:

- $u_{-1} \leftarrow h, u_0 \leftarrow s, v_{-1} \leftarrow 0, v_0 \leftarrow 1$;
- 对 $i = 0, 1, \ldots$ 执行

  $$q_i = \lfloor u_i/u_{i-1} \rfloor, u_{i+1} = u_{i-1} - q_i u_i, v_{i+1} = v_{i-1} + q_i v_i,$$

  直到 $u_k \leq h^{1/2}$. 则

  $$f/g = (-1)^k u_k/v_k.$$

在实用中, 对于连分式展式部分可以采用 Lehmer 加速算法, 可以参考 [104]. 经过详细的分析, Dixon 指出这种算法的复杂度为 $O(n^3(\log n)^2)$, 接近数值算法的渐近复杂度 $O(n^3)$, 而优于采用基于中国剩余定理的模算法的算法复杂度.

6.2.4 数值算法求精确解

与 Hensel 提升方法类似的另一种思路是将消元步骤化归为机器精度的浮点数运算. 由于现代计算机往往具有很强的浮点数运算能力, 通过将高精度的整数通过一定的近似手段化归为机器精度的浮点数运算可以大大提高计算的效率. 我们在下面给出的这一算法 [176], 对于适当良态的非奇异矩阵, 通过近似与迭代步骤得到足够精度的浮点数解后, 通过连分式展开的方法恢复有理数解. 而对于病态的矩阵, 则很快判断并退出执行. 相较于 Wan 给出的原始算法有效性证明, 下面给出的证明充分利用了连分式展开的性质, 因而更简洁.

对于良态矩阵解的迭代逼近, 在 [76] 中有一些介绍. 然而, 这种方法并没有将全部计算化归为单精度(或者说, 机器精度)的计算, 特别在将历次修正累加时, 仍需要利用高精度的计算. 因此, 对我们目前的要求来说还不够. 我们的迭代计算过程如下:对于输入非奇异整系数矩阵 A 与整数向量 b, 将解初始化为 $x = n/d$, 其中 $n = 0$, $d = 1$, 将余量 r 置为 b. 随后在每次迭代中, 执行如下的步骤:在机器精度运算中找到近似解 Δx 满足 $A\Delta x = r$, 选择合适的量 α, 对方程进行放大, 即令 $\Delta d = \alpha$, $\Delta n = (\approx \Delta\alpha \cdot x_1, \cdots, \approx \Delta\alpha \cdot x_n)$, 将方程解修正为 $n = \alpha n + \Delta n$, $d = \Delta d \cdot d$, 余量修正为 $r = \alpha r - A \cdot \Delta n$. 这样, 在每步迭代步骤之后, 出现在结果中的数, 包括近似解的分子与分母以及(扩大后的)余量, 都是不超过 $\|A\|_\infty + \|b\|_\infty$ 的整数. 从而, 迭代过程中我们不需要高精度运算. 当近似解达到足够精度之后, 我们进行如下算法中叙述的过程得到方程组的精确解.

算法6.6 (数值方法求稠密线性方程组有理解).

输入:A 为 $m \times m$ 非奇异整系数矩阵, b 为整系数向量.

输出:当 A 为良态矩阵时, 快速输出方程 $Ax = b$ 的有理解 x;否则, 快速退出并输出“数值精度不足”的信息.

1. 使用机器精度的浮点运算得到 A 的 LUP 分解(其他分解也可使用).
2. 置解向量的公分母 $d^{(0)} = 1$.
3. 置余向量 $r^{(0)} = 0$.
4. 置计数器 $i = 0$.
5. 计算 A 的 Hadamard 界.
6. 重复执行以下步骤, 直到 $d^{(i)} > 2mB^2\left(2^{-i}\|b\|_\infty + \|A\|_\infty\right)$:

- i:=i+1.
- 利用第 1 步得到的 LUP 分解, 用浮点运算计算 $\bar{x}^{(i)} = A^{-1}r^{(i-1)}$.
- 计利用浮点运算算 $\alpha^{(i)} := \min\left(2^{30}, 2^{\lfloor log_2(\frac{\|r^{(i-1)}\|_\infty}{\|r^{(i-1)} - A\bar{x}^{(i)}\|_\infty})-1\rfloor}\right)$.
- 若 $\alpha^{(i)} < 2$, 则退出并提示信息“数值精度不足”.
- 严格计算整系数向量 $x^{(i)} := (\approx \alpha^{(i)} \cdot \bar{x}_1^{(i)}, \cdots, \approx \alpha^{(i)} \cdot \bar{x}_m^{(i)})$, 也即使得 $x^{(i)}$ 为满足 $\|x^{(i)} - \alpha^{(i)} \cdot \bar{x}^{(i)}\|_\infty \leq 0.5$ 的向量.
- 严格计算整数 $d^{(i)} := d^{(i-1)} \cdot \Delta\alpha^{(i)}$.
- 严格计算整系数向量 $r^{(i)} := \alpha^{(i)} \cdot r^{(i-1)} - Ax^{(i)}$, 这时余量扩大了 $d^{(i)}$ 倍.

7. 置 $k := i$.
8. 计算整系数向量 $n^{(k)} := \sum\limits_{1\leq i\leq k} \frac{d^{(k)}}{d^{(i)}} \cdot x^{(i)}$, 注意到 $\frac{d^{(k)}}{d^{(i)}} = \prod\limits_{i<j\leq k} \alpha^{(j)}$.
9. 此时, 精确解 x 构成 $\frac{1}{d^{(k)}} \cdot n^{(k)}$ 的一个最佳逼近, 且其分母有上界 B. 可利用连分式展开的方法得到 x.
10. 返回 x.

注100. 第 6 步中 $\alpha^{(i)}$ 的选择是基于如下考虑:选择 2 的幂次使得以下的计算变为简单的移位过程, 从而更为高效;若 A 为良态矩阵, 则余量可能已经非常小, 必须加一个限制以保证以下的乘法不会溢出, 这一限制被选择为 2^{30} 同样是为了保证计算结果在机器精度之内.

首先我们来估计每次迭代过后余量的上界, 并由此导出以上算法的正确性.

定理6.9 (算法的正确性). *若以上算法中每步计算得到的 $\alpha^{(i)}$ 都有*

$$\alpha^{(i)} \leq \frac{\|r^{(i-1)}\|_\infty}{2\|r^{i-1} - A\bar{x}^{(i)}\|_\infty}, \tag{6.10}$$

则以上算法将正确执行, 即正常地由于矩阵病态而退出, 或者得到正确的结果. 且在第 i 步迭代中

$$\|r^{(i)}\|_\infty = \|d^{(i)}(b - A\frac{1}{d^{(i)}} \cdot n^{(i)})\|_\infty \leq 2^{-i}\|b\|_\infty + \|A\|_\infty.$$

证明. 对于输入 A 与 b, 我们只需要证明若每次计算出的 $\alpha^{(i)} \geq 2$, 则算法能够顺利执行完毕并得到正确结果. 此时, 由

$$d^{(i)} = \prod_{1 \leq j \leq i} \alpha^{(j)} \geq 2^i$$

可知循环步骤只能执行有限次而最终退出.

记 $n^{(i)} = \sum\limits_{1 \leq j \leq i} \frac{d^{(i)}}{d^{(j)}} \cdot x^{(j)}$ 为 i 步迭代之后解的分子. 我们要估计绝对误差 $\|e^{(i)}\|_\infty = \|\frac{1}{d^{(i)}} \cdot n^{(i)} - A^{-1}b\|_\infty$, 由归纳法容易得到

$$r^{(i)} = d^{(i)}(b - A\frac{1}{d^{(i)}} \cdot n^{(i)}),$$
$$e^{(i)} = \|\frac{1}{d^{(i)}} \cdot n^{(i)}\|_\infty = \frac{1}{d^{(i)}}\|A^{-1}r^{(i)}\|_\infty.$$

在每步迭代中, 根据定理的假设, $\|A\bar{x}^{(i)} - r^{(i-1)}\|_\infty \leq \frac{1}{2\alpha^{(i)}} \cdot \|r^{(i-1)}\|_\infty$. 根据 $x^{(i)}$ 的定义, 我们有 $\|x^{(i)} - \alpha^{(i)}\bar{x}^{(i)}\|_\infty \leq 0.5$. 从而,

$$\begin{aligned}\|r^{(i)}\|_\infty &= \|Ax^{(i)} - \alpha^{(i)} \cdot r^{(i-1)}\|_\infty \\ &\leq \|\alpha^{(i)} \cdot A\bar{x}^{(i)} - \alpha^{(i)} \cdot r^{(i-1)}\|_\infty + \|Ax^{(i)} - \alpha^{(i)} \cdot A\bar{x}^{(i)}\|_\infty \\ &\leq \frac{1}{2}\|r^{(i-1)}\|_\infty + \frac{1}{2}\|A\|_\infty,\end{aligned}$$

据此, 可以得到

$$\|r^{(i)}\|_\infty \leq \frac{1}{2^i}\|b\|_\infty + \|A\|_\infty.$$

误差

$$e^{(i)} = \frac{1}{d^{(i)}}\|A^{-1}r^{(i)}\|_\infty \leq \frac{1}{d^{(i)}}\|A^{-1}\|_\infty \left(\frac{1}{2^i}\|b\|_\infty + \|A\|_\infty\right), \forall i \leq 1.$$

设 k 为循环停止时计数值, 则

$$2B\det(A)e^k < \frac{2}{d^{(k)}}\|B\det(A) \cdot A^{-1}\|_\infty(2^{-k}\|b\|_\infty + \|A\|_\infty).$$

根据 Cramer 法则, $\det(A)A^{-1}$ 正是 A 的古典伴随矩阵, 它的每个元素都是 A 的 $m-1$ 阶子式, 从而,

$$2B\det(A)e^{(k)} \leq \frac{2mB^2(2^{-k}\|b\|_\infty + \|A\|_\infty)}{d^{(k)}} < 1.$$

这样就得到 $e^{(k)} < \frac{1}{2B\det(A)}$, 也即 $\|\frac{n^{(k)}}{d^{(k)}} - A^{-1}b\|_\infty < \frac{1}{2B\det(A)}$. 同样根据 Cramer 法则, 我们知道 $\det(A)A^{-1}b$ 为整系数向量. 于是根据定理 6.8 , 知道 $A^{-1}b$ 构成 $\frac{n^{(k)}}{d^{(k)}}$ 的一个最佳逼近, 从而可以由 $\frac{n^{(k)}}{d^{(k)}}$ 的连分式展开得到 $A^{-1}b$. 这就完成了证明. □

在利用近似解恢复精确有理解的阶段, 可以采用 [179] 中建议的一种方法, 可以概括为:首先利用概率性算法得到矩阵 A 的最大的不变因子(定义见下, 也可见 [12])或它的一个因子, 随后可以更快地计算上述恢复过程.

定理6.10 (整系数矩阵的 Smith 标准形). *设 A 为 $\mathbb{Z}$ 上的矩阵, 秩 $\operatorname{rank}(A)=r$. 存在 $\mathbb{Z}$ 上的可逆方阵 P 和 Q, 使得*

$$B=PAQ=\begin{bmatrix} s_1 & & & & & \\ & \ddots & & & & \\ & & s_r & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{bmatrix},$$

其中 $s_i|s_{i+1}$, $i=1,\cdots,r-1$. $(s_1,\cdots,s_r)$ 称为 A 的不变因子组, B 称为 A 的 Smith 标准形.

证明. 证明可参见 [12] 定理 7.13. □

注101. 为了使如上定义的 Smith 标准形唯一, 常常规定整系数矩阵的不变因子 $s_i\geq 0, 1\leq i\leq r$. 以下我们将采用这一规定.

定理6.11. *对于 $m\times m$ 阶非奇异非奇异方阵 A, 设其最大不变因子为 s_n, 则 s_nA^{-1} 为整系数矩阵.*

证明. 设 $B=PAQ$ 为 A 的 Smith 标准形, 则 s_nB^{-1} 为整系数矩阵, 而 P, Q 为整系数可逆方阵, 从而 $s_nA^{-1}=Q(s_nB^{-1})P$ 为整系数方阵. □

根据该定理, 若 s_n 已知, 则算法 6.6 中由近似解恢复精确解成为平凡的, 因为这时 $s_nA^{-1}b$ 为整系数向量, 从而在计算足够精度之后将近似解截断即可. 即使仅能得到 s_n 的一个非平凡因子 s_n^*, 将其乘到近似解 $\frac{n^{(k)}}{d^{(k)}}$ 上, 就只要计算 $s_n^*\frac{n^{(k)}}{d^{(k)}}$ 的一个分母不超过 B/s_n^* 的最佳逼近即可. 下面给出随机性地计算 A 最大不变因子的算法.

算法6.7 (最大不变因子).

输入:n 阶非奇异整系数矩阵 A.

输出:正整数 $s_n^*|s_n$.

1. (初始化)$M := \max\{\lceil\sqrt{n\log\|A\|_\infty}\rceil, 4000\}$, p 为约为 $n\log A$ 的素数, $b^{(1)}, b^{(2)}, c^{(1)}, c^{(2)}$ 为随机生成的 n 维整系数列向量. $h := \lceil 2\log_p(\|A\|_\infty^{2n-1}M)\rceil$, 从而 $q := p^h \geq \|A\|_\infty^{2n-1}M$.
2. 对 $k = 1, 2$, 执行如下计算步骤:
   - $x^{(k)} = (x_i^{(k)})_{i=1}^n := A^{-1}b^{(k)} \in F_q^n$.
   - $y^{(k)} := c^{(k)T}x^{(k)} = \sum_{i=1}^n c_i^{(k)}x_i^{(k)} \in F_q$.
   - 取 $t_n^{(k)}$ 为 $y^{(k)}$ 的分母, $t_n^{(k)} := \delta(y^{(k)})$, 从而 $0 \leq t_n^{(k)} \leq \|A\|_\infty^n$.
3. 输出 $s_n^* := \mathrm{lcm}(t_n^{(1)}, t_2^{(2)})$.

定理6.12 (算法的有效性). 算法 *6.7* 中, $s_n^*|s_n$, 且能以不小于 $1/2$ 的概率给出 $s_n^* = s_n$ 的结果.

为了证明 6.7 算法的有效性, 首先证明如下引理, 它给出了以上随机化过程带来的结果. 为了叙述方便, 我们引入如下记号:对整数 x 与素数 p, 定义 $g = \mathrm{ord}_p x$ 满足 $p^g|x$, $p^{g+1}\nmid x$, 称为 x 关于 p 的阶. 用 P 表示概率.

引理6.2. 记 $\delta^{(k)} := \mathrm{lcm}(\delta(x_1^{(k)}), \cdots, \delta(x_n^{(k)}))$, 显然 $t_n^{(k)}|\delta^{(k)}$. 则对任意素数 $\bar{p}$, 有结论如下:

1. $P(\mathrm{ord}_p s_n \neq \mathrm{ord}_p \delta^{(k)}) \leq \max\{1/M, 1/p\}$;
2. $P(\mathrm{ord}_p t_n^{(k)} \neq \mathrm{ord}_p \delta^{(k)}) \leq \max\{1/M, 1/p\}$.

该引理证明参见 [179].

算法有效性的证明. $s_n^*|s_n$ 根据计算过程容易归纳得到. 只需证明后一论断.

$$\begin{aligned} P(s_n \neq s_n^*) &\leq \sum_{\bar{p}|s_n, \bar{p}\text{ prime}} P(\mathrm{ord}_{\bar{p}} s_n^* < \mathrm{ord}_{\bar{p}} s_n) \\ &= \sum_{\bar{p}|s_n, \bar{p}\text{ prime}} P(\mathrm{ord}_{\bar{p}} t_n^{(1)} < \mathrm{ord}_{\bar{p}} s_n \wedge \mathrm{ord}_{\bar{p}} t_n^{(2)} < \mathrm{ord}_{\bar{p}} s_n) \\ &\leq \sum_{\bar{p}|s_n, \bar{p}\text{ prime}} (\max\{1/M, 1/\bar{p}\})^2 \\ &< \frac{1}{2}. \end{aligned}$$

□

在算法 6.7 中, 我们仍然需要求解线性方程组, 这是不是陷入了一种循环呢?注意到, 在算法中, 我们只需在 F_q 上求解两个线性方程组, 而 $y^{(k)}, k = 1, 2$ 的恢复只需要少量运算, 而不像之前算法中需要 n 次恢复运算. 这样, 当 n 很大时, 我们可以利用这种方法进行加速.

注102. 由定义可知, $\alpha^{(i)}$ 大致只与矩阵 A 的条件数有关. 因此, 我们可以只计算一次 α, 而将之后的计算中取 α 相同. 这可能造成得到的 α 不满足定理 (6.9) 中 (6.10) 式的要求. 然而, 我们可以通过检查余量的范数是否达到理论预测的值来发现这种情形. 每当这种情形发生时, 我们将 α 减半, 再次代入运算即可.

注103. 算法 6.6 的数值求解算法不限于 LUP 分解. 任何具有一定数值稳定性的算法都可以用来求近似解, 这种算法可以包括稠密线性方程组的多种分解方法以及稀疏线性方程组的迭代算法等, 可参考 [76]. 在 [176] 一文中就利用这种思想构造了一个稀疏线性方程组的求解算法.

注104. 如果不要求精确解的话, 算法 6.6 不需要执行完毕, 而仅作为一个可扩展精度的数值型求解算法, 它具有较高的执行效率并能有效地估计解的精度.

注105. [176] 指出, 这一算法的渐近复杂度与 Dixon p-adic 算法相同, 均为 $O^{\sim}(m^3)$ 阶的, 其中 $\sim$ 符号表示忽略了一个可能含有的 $\log m$ 的幂次, 也可记为 $O(m^{3+\epsilon})$ 阶. 但实际测试表明, 本算法的执行效率比 p-adic 算法好很多.

6.3 Wiedemann 算法与黑箱方法

在数值线性代数中, 黑箱(black box)算法已经广为人知. 这一类算法的基本特点是, 在算法执行过程中, 不涉及对矩阵元素的直接操作(例如直接的消元步骤), 矩阵的作用仅体现为能够对向量施行线性变换, 体现为一个“黑箱”. 在数值运算中, 黑箱方法往往与迭代法紧密相连, 通过迭代操作使中间结果快速收敛. 对于稀疏矩阵或有结构的矩阵(如 Vandermonde 阵或 Toeplitz 阵), 由于存在黑箱作用的快速算法, 在作用过程中不会破坏矩阵的稀疏性或结构性质, 因而与消元法等直接算法相比, 能够大大减少计算量. 关于数值计算中的黑箱算法, 读者可以参考 [76][169].

自 1986 年 Wiedemann[184] 将黑箱算法用于计算有限域上的稀疏线性方程组的精确解以来, 精确线性代数中的黑箱型算法已经得到了很大的发展. 针对不同的问题, 出现了大量高效的黑箱算法;另一方面, 随机化预处理步骤在精确计算中得到广泛运用. 在本节中, 我们将首先简述概率性算法与预处理步骤的概念, 随后着重介绍 Wiedemann 算法[184].

6.3.1 概率性算法与预处理步骤概述

在传统的计算中, 算法往往是“确定性”的, 即只要能够正确地按照算法步骤执行, 总能得到正确的结果. 但当我们试图考虑一些复杂的问题时, 要求“万无一失”的确定性算法往往复杂性很高, 难于应用. 而另一方面, 我们总能够构造一些算法, 它们能够高效地执行, 并以一定的概率得到正确的结果, 这就是我们所称的概率性算法. 我们在设计这种算法的同时, 必须给出得到正确结果的概率下界, 从而了解这种算法的可靠性. 在“素性判定”一节中, 我们讲过很多这样的算法, 如 Solovay-Strassen 算法等. 上一节中的最大不变因子求解过程中也采用了随机化的步骤. 这里, “一定的概率”可能体现为两种情形:一是算法总能快速地执行完毕, 但可能得不到正确的结果, 而是返回错误的结果或者只返回一条警告信息, 这称为 Monte Carlo 型算法;二是当算法执行完毕时总能返回正确的结果, 但是并非总能快速地执行, 而只能在平均意义下或对一些好的情形快速给出结果, 这称为 Las Vegas 型算法.

注意到, 以上两种类型算法的区别并不是绝对的. 比如说, 如果我们能够快速检验计算结果的正确性(如整数因子分解中的乘法检验), 则我们只要反复执行该算法, 直到我们得到一个正确结果, 则该算法成为 Las Vegas 型的. 而对于 Las Vegas 型算法, 如果我们发现程序执行了过长的时间而没有返回结果, 我们令算法中止同时返回一个警告信息(随后可以重开一个随机化步骤执行算法或者跳转到其他的算法), 这时我们可以认为它成为了一个 Monte Carlo 型的算法.

那么, 为什么算法会带有随机的性质呢?这是因为算法中包含了随机化的步骤. 在线性代数领域, 这种随机化步骤往往是在中间步骤中通过随机产生向量或矩阵进行运算实现的. 对于黑箱算法以及更广泛的算法而言, 最重要的随机化过程称为预处理步骤, 这是指对输入矩阵 A 施行一个带有随机性的变换 $\mathcal{P}: A \longmapsto A'$, 使得经过预处理后 A' 具有一定的代数性质, 能够满足进一步运算的要求. 由于这种变换存在的随机性, 其结果可能并不满足要求, 这往往体现为它具有一定的奇异性. 为了控制这种奇异情形的概率, 我们往往通过与有关多项式的零点数的 Schwartz-Zippel 定理来做估计.

接下来我们介绍上面提到的 Schwartz-Zippel 定理[156].

定理6.13 (Schwartz-Zippel). *设 $Q \in F[x_1, \cdots, x_n]$, $Q \neq 0$ 是 n 元多项式. 设 Q_1 是 Q 按照 x_1 的降幂排列得到的多项式, d_1 为 x_1 在 Q_1 中的次数, Q_2 为 $x_1^{d_1}$ 在 Q_1 中的系数多项式. 归纳地说, 令 d_i 为 Q_i 中 x_i 的次数, Q_{i+1} 为 Q_i 中 $x_i^{d_i}$ 的系数, $1 \leq i \leq n$. 对 $1 \leq i \leq n$, 令 x_i 在 $I_i \subseteq F$ 中取值, 则在 $I_1 \times \cdots \times I_n$ 中 Q 至多有*

$$|I_1 \times \cdots \times I_n| \left(\frac{d_1}{|I_1|} + \cdots + \frac{d_n}{|I_n|} \right)$$

个零点.

这一定理给出了多元多项式在其定义集合上零点个数的上界. 在我们采取的随机化步骤中, 往往需要在某个集合中随机取值构造向量或矩阵. 在此过程中, 可能导致算法失败的奇异情形往往表现为取到了某个多元多项式的零点. 根据 Schwartz-Zippel 定理, 我们就可以对此失败的概率做出上限估计.

证明. 归纳法. $n=1$ 的情形由代数基本定理可得.

设以上命题对 $Q_2 \in F[x_2,\cdots,x_n]$ 成立, 即其零点集 $\mathrm{Null}(Q_2)$ 有 $\#\mathrm{Null}(Q_2) \leq |I_2\times\cdots\times I_n|(\frac{d_2}{|I_2|}+\cdots+\frac{d_n}{|I_n|})$. 若 $(z_2,\cdots,z_n)\in\mathrm{Null}(Q_2)$, 则 $\forall x_1\in I_1$, $(x_1,z_2,\cdots,z_n)\in I_1\times\cdots\times I_n$ 都可能是 $Q_1\in F[x_1,\cdots,x_n]$ 的零点. 若 $(z_2,\cdots,z_n)\notin\mathrm{Null}(Q_2)$, 则至多有 d_1 个 $x_1\in I_1$ 使得 $(x_1,z_2,\cdots,z_n)\in\mathrm{Null}(Q_1)$. 从而 Q_1 在 $I_1\times\cdots I_n$ 中至多有

$$\begin{aligned}&|I_1||I_2\times\cdots\times I_n|\left(\frac{d_2}{|I_2|}+\cdots+\frac{d_n}{|I_n|}\right)+d_1|I_2\times\cdots\times I_n|\\=&|I_1\times I_n|\left(\frac{d_1}{|I_1|}+\cdots+\frac{d_n}{|I_n|}\right).\end{aligned}$$

个零点. □

Schwartz-Zippel 定理有以下直接的推论, 对于概率性算法的分析有重要意义.

推论6.1. *设 $Q\in F[x_1,\cdots,x_n]$, $Q\neq 0$, 且 $x_1,\cdots,x_n$ 均在 $U\subseteq F$ 中随机选取. Q 结果为零的概率不超过 $\frac{D}{\#U}$, 其中 D 为 Q 的次数.*

推论6.2. *若 $F=\mathbb{R}$ 或 $F=\mathbb{C}$, $Q\in F[x_1,\cdots,x_n]$, $Q\neq 0$. 若 $(x_1,\cdots,x_n)$ 在非零测集 $U\subseteq F$ 上随机选取, 则 $Q(x_1,\cdots,x_n)$ 在概率 1 的意义下非 0.*

下面, 我们通过几个例子说明预处理步骤的应用. 关于线性代数中涉及到的更广泛的预处理问题以及相应的预处理方法, 读者可以参考 [51].

分块 Gauss 消元法

我们已经熟悉带选主元的 Gauss 消元法. 理论上, 我们每个消元步骤也可不限于单个元素, 考虑如下的块消元过程:

$$\begin{bmatrix}A & B\\ C & D\end{bmatrix}\begin{bmatrix}I & -A^{-1}B\\ O & I\end{bmatrix}=\begin{bmatrix}A & O\\ C & D-CA^{-1}B\end{bmatrix}.$$

这样的消元有如下好处:它借助分块矩阵计算可将消元化为递归的过程, 这有利于算法复杂度的降低;同时, 经过化归后, 算法容易并行化. 然而, 这样的算法应用于

一般矩阵时将遭遇重大的困难:算法步骤中要求 A 必须可逆, 而这并非普遍得到满足的. 事实上, 按元素进行的 Gauss 消元法必要的选主元过程, 正是为了保证主子式的可逆性, 从而能够顺利执行消元步骤.

如果对系数矩阵 M 进行预处理:

$$\mathcal{P} : M \longmapsto M'x = UMV,$$

其中 U 和 V 为在一定范围内随机选取的矩阵, 使得 M' 满足主子式非奇异, 则块消元过程可以执行下去.

例6.1 (随机四分变换). 若

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

为 $(2n)\times(2n)$ 阶非奇异实矩阵, $R = \mathrm{diag}(r_1, \ldots, r_n)$, $S = \mathrm{diag}(s_1, \ldots, s_n)$ 为 $n \times n$ 阶对角阵, 其元素在 $U \subseteq \mathbb{R}$ 中随机选取. 则

$$M' = \begin{bmatrix} \tilde{A} & \tilde{B} \\ \tilde{C} & \tilde{D} \end{bmatrix} = \begin{bmatrix} I & R \\ I & -R \end{bmatrix} \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} I & I \\ S & -S \end{bmatrix}$$

的任意阶主子阵均以概率 1 非奇异.

证明. 可以证明, 以上得到的 M' 每一主子阵的行列式均可表示为 $(r_1, \cdots, r_n)$ 和 $(s_1, \cdots, s_n)$ 的非零多项式, 由推论得到对它进行计值时以概率 1 非 0. □

以上的随机四分变换是一类更普遍的随机蝶形变换(random butterfly transformation)的特例. 这类变换的思想来源于一类复杂网络问题. 由于随机四分矩阵相当稀疏, 其乘法计算量很小, 因此它提供了一个很方便的分块矩阵预处理器, 可以参考 [136].

类似地, 下面给出的 Toeplitz 预处理步骤也可达到同样的效果 [96].

例6.2. 设 F 为域(通常为有限域). $M \in F^{n\times n}$, 取 F 中的子集 $S \subset F$. 考虑如下的线性变换:

$$M' := UMV^T,$$

其中

$$U := \begin{pmatrix} 1 & u_2 & u_3 & \cdots & u_n \\ & 1 & u_2 & \cdots & u_{n-1} \\ & & 1 & \ddots & \vdots \\ & & & \ddots & u_2 \\ & & & & 1 \end{pmatrix},$$

$$V := \begin{pmatrix} 1 & v_2 & v_3 & \cdots & v_n \\ & 1 & v_2 & \cdots & v_{n-1} \\ & & 1 & \ddots & \vdots \\ & & & \ddots & v_2 \\ & & & & 1 \end{pmatrix},$$

未定元 $u_2, \ldots, u_n, v_2, \ldots, v_n$ 从 S 中随机选取. 记 $r = \mathrm{rank}(A)$, 记 A'_i 为 A' 的 i 阶前主子阵(leading principal minors), 则

$$\mathrm{Prob}(\det(A'_i) \neq 0, \forall 1 \leq i \leq r) \geq 1 - \frac{r(r+1)}{\#S}.$$

证明参考 [96]. 由于 Toeplitz 阵与向量的乘法等价于两个多项式的乘法, 可用 $O^{\sim}(n)$ 步计算得到, 从而在对 M 施行如上预处理步骤后, 得到的 M' 仍是一个好的黑箱矩阵.

一个概率性的求秩算法

秩是矩阵的一个重要的不变量, 许多线性代数的算法都假定矩阵的秩已知. 通常的求秩算法是以 Gauss 消元法为基础的. 为了适应黑箱算法的应用, 下面给出一个基于极小多项式计算的概率性的矩阵求秩算法 [96].

我们知道, 对于一般矩阵 M 而言, 由于特征值的简并性质, 秩 r 与极小多项式的次数 m_A 没有直接的关系. 但如果矩阵的特征多项式除 0 以外没有重根, 且矩阵的 Jordan 标准形中没有超过 1 阶的幂零 Jordan 块(即 0 的代数重数等于其几何重数 [12]), 则总有 $r = m_M - 1$. 下面给出的对角阵预处理步骤事实上将矩阵转化为这种情形. 证明同样参见 [96].

例6.3. 设 $M \in F^{n\times n}$ 具有直到 r 阶的非奇异前主子阵, 其中 $r = \mathrm{rank}(M)$(未知), 并设 $r < n$. 令 $X = \mathrm{diag}(x_1, \cdots, x_n)$, 其中 $x_1, \cdots, x_n$ 在子集 $S \subset F$ 中随机选取, 记 $M' = MX$, 则

$$\mathrm{Prob}(r = \deg(m_{M'}) - 1) \geq 1 - \frac{n(n-1)}{\#S}.$$

对一个一般的方阵, 在进行如上两个预处理步骤之后, 利用后面两节中介绍的矩阵极小多项式算法即可得到矩阵的秩. 对于稀疏或有结构的矩阵, 这一概率性算法的效率要高于直接的消去法.

从以上例子中我们可以归纳出预处理器的构造思路:

1. 预处理过程即对矩阵 $M \in F^{m\times n}$ 进行如下的线性变换 $\mathcal{P} : M \longmapsto PMQ$, $P \in F[x_1, \cdots, x_s]^{m\times m}$, $Q \in F[y_1, \cdots, y_t]^{n\times n}$.

2. 其中 $x_1,\cdots,x_s,y_1,\cdots,y_t$ 在 F(或 $U\subseteq F$)上随机选取.

3. 由于要证明的性质对应于 $F[x_1,\cdots,y_t]$ 上非零多项式, 因此变换后的矩阵在一定概率下满足我们要求的性质.

在构造 P, Q 的过程中, 有一些原则是要遵守的:

- P, Q 的选取要尽量简单.
- P, Q 要具有好的性质, 如易于求逆, 能够快速施行矩阵-向量乘法等.

关于预处理步骤的全面的论述, 请读者参阅 [51].

6.3.2 线性递推列

在本节中, 我们首先介绍线性递推序列(linear recurrent sequences)及其极小多项式的概念以及相关算法.

定义6.3 (线性递推列). 设 F 为域, V 为 F 上的线性空间, 记 V 上的无穷序列 $(a_i)_{i\in\mathbb{N}}, a_i\in V, i\in\mathbb{N}$ 全体构成的线性空间为 $V^{\mathbb{N}}$. 称 $a=(a_i)\in V^{\mathbb{N}}$ 为线性递推列, 若存在 $n\in\mathbb{N}$ 及 $f_0,\cdots,f_n\in F, f_n\neq 0$, 使得

$$\sum_{0\le j\le n} f_j a_{i+j} = f_n a_{i+n}+\cdots+f_1 a_{i+1}+f_0 a_i = 0$$

对于任意的 $i\in\mathbb{N}$ 都成立. n 阶多项式 $f\equiv\sum_{0\le j\le n} f_j x^j\in F[X]$ 称为 a 的特征多项式, 也称为零化多项式或生成多项式.

根据以上定义, 将 F 和 V 均取为 $\mathbb{Q}$, 则我们熟悉的 Fibonacci 序列是线性递推列, x^2-x-1 构成它的一个特征多项式. 我们知道, 对于一个线性递推列, 在已知其特征多项式及最初的几个取值时, 可以通过递推的方法快速计算其后继序列. 下面我们举一些线性代数中涉及的线性递推列的例子, 它们在下面的算法中将扮演重要角色.

例6.4. 本例给出线性代数中一些重要的线性递推列. 这里最关键的是利用了线性代数中的 Caley-Hamilton 定理 [12]. 以下 F 均表示一般域.

1. 取 $V=F^{n\times n}$, $A\in V$ 为任意方阵. 则 $a=(A^i), i\in\mathbb{N}$ 为线性递推列, A 的极小多项式与特征多项式都是 a 的特征多项式.

2. 设 $V=F^n$, $A\in F^{n\times n}$, $b\in F^n$ 为任意向量. 注意到 (A^i) 的任意特征多项式也将 (A^ib) 化为零, 故 (A^jb) 也为线性递推列. 由 a 的元素生成的 V 中的线性子空间称为 A 与 b 的 Krylov 子空间.

3. 设 $V = F^n$, $A \in F^{n\times n}$, $b, u \in F^n$ 为 V 中的任意向量. 则 $(A^i b)$ 的任意特征多项式同样将 $(u^T A^i b)$ 化为零, 故 $(u^T A^i)b$ 也为线性递推列.

我们可以通过引入多项式对序列的作用, 将无穷序列与多项式更紧密地联系起来.

定义6.4 (多项式对序列的作用). 设 $f = \sum_{0\le j\le n} f_j x^j \in F[X]$ 以及序列 $a = (a_i)_{i\in\mathbb{N}} \in V^{\mathbb{N}}$, 定义 f 对 a 的作用如下

$$f \bullet a = \left(\sum_{0\le j\le n} f_j a_{i+j} \right)_{i\in\mathbb{N}} \in V^{\mathbb{N}}.$$

注意到, 在此定义下, 常系数对序列的作用即数乘作用, 而未定元 x 对 a 的作用相当于平移算子. 在此作用下, $V^{\mathbb{N}}$ 成为 $F[X]$ 上的模.

固定 $a \in V^{\mathbb{N}}$, 容易验证 a 的特征多项式的集合与 0 一起, 构成了 $F[X]$ 中的理想, 我们称之为 a 的零化子理想, 记为 $\mathrm{Ann}(a)$. 由于 $F[X]$ 为主理想整环, 记 $\mathrm{Ann}(a)$ 首项系数为一的生成元为 m_a, 则 m_a 的倍数全体构成了 a 的特征多项式集合, m_a 是其中次数最低的首一多项式, 我们称之为 a 的极小多项式. m_a 的次数称为 a 的递推阶数. 结合例 6.4 中的几个例子, 我们可以对线性代数中的特征多项式, 极小多项式等概念有更深刻的理解.

本节的主要目的是给出线性递推序列的极小多项式的构造性算法. 以下定理给出了 $V = F$ 时线性递推列的极小多项式的一个判别法则.

定义6.5 (多项式的倒逆). 设 $f(x) = f_d x^d + \cdots + f_0 \in F[X]$ 为 d 阶多项式, 记 $\mathrm{rev}(f) \equiv x^d f(x^{-1}) = f_0 x^d + \cdots + f_d \in F[X]$, 称为 f 的倒逆.

定理6.14. *设 $a = (a_i)_{i\in\mathbb{N}} \in F^{\mathbb{N}}$ 为线性递推序列, $h = \sum_{i\in\mathbb{N}} a_i x^i \in F[\,[X]\,]$, $f \in F[X]\backslash 0$, $\deg f = d$, $r = \mathrm{rev}(f)$.*

1. *以下命题等价:*
   - *f 为 a 的特征多项式;*
   - *r, h 为低于 d 阶的多项式;*
   - *$h = g/r$, 其中 $g \in F[X]$, 且 $\deg g < d$.*
2. *若 f 为 a 的极小多项式, 则 $d = \max\{1 + \deg g, \deg r\}$, 且 $\gcd(g, r) = 1$.*

证明. 1. 直接代入按照形式幂级数的乘法规则验证即可.

2. $d \geq \deg r$, 故 $d \geq \max\{1+\deg g, \deg r\}$. 若 $d > \deg r$, 则 $x|f$, 且 f/x 为 a 的特征多项式, 与 f 极小矛盾. 设 $u = \gcd(g, r)$, 则 $f^* \equiv f/\mathrm{rev}(u)$ 为 $d - \deg r$ 次多项式, $r/u = \mathrm{rev}(f^*)$, 且 $(r/u)\cdot h = g/u$ 为 $\deg g - \deg u$ 次多项式. 于是 r/u 为 a 的特征多项式, $u \in F$.

□

在域 F 上, 若给定一个线性递推列 $a \in F^{\mathbb{N}}$, 且我们能够预先知道其极小多项式 m_a 的次数上界. 例如在前面的例 6.4 的 3 中, m_a 的最高次数不会超过矩阵 A 的阶数. 在这种情况下, m_a 可以通过求解如下的 Padé 逼近问题得到解答:

$$h \equiv \frac{s}{t} \pmod{x^{2n}}, x \nmid t, \deg s < n, \deg t \leq n, \gcd(s,t) = 1. \tag{6.11}$$

我们看到, $(s,t) = (g,r)$ 正是如上问题的一组解. 由 Padé 逼近问题解的唯一性, 即得如下算法.

算法6.8 (Berlekamp-Massey)**.**

输入:域 F 上线性递推列 $a \in F^{\mathbb{N}}$, 给定 a 的递推阶数的一个上界 n 以及 a 的前 $2n$ 个元素 $a_0, \cdots, a_{2n-1} \in F$.

输出:a 的极小多项式 $m_a \in F[X]$.

1. $h := a_{2n-1}x^{2n-1} + \cdots + a_1 x + a_0$, 使用扩展 Euclid 算法计算 $s, t \in F[X]$ 且 $t(0) = 1$, 使得 (6.11) 成立.
2. $d := \max\{1 + \deg s, \deg t\}$.
3. 返回 $\mathrm{rev}_d(t) = x^d t(1/x)$.

经过分析, 这一算法能在 $O(n^2)$ 的域 F 运算步骤内给出 a 的极小多项式.

6.3.3 线性方程组的 Wiedemann 算法

下面我们考虑线性方程组 $Ax = b$, 其中系数矩阵 $A \in F^{n\times n}$ 非奇异, 则该线性方程组有唯一解. 设 $m = m_b = \sum_{0\leq j\leq d} m_j x^j \in F[X]$ 为线性递推序列 $a_b = (A^i b)_{i\in\mathbb{N}}$ 的极小多项式, 则 $m \bullet a = m(A)b = \sum_{0\leq j\leq d} m_j A^j b = 0$, 从而我们得到

$$A \cdot (-m_0^{-1}) \sum_{1\leq j\leq d} m_j A^{j-1} b = b,$$

即 $x = -m_0^{-1}\sum_{1\leq j\leq d} m_j A^{j-1} b$ 为方程组的解. 据此分析, 我们得到如下算法:

算法6.9.

给定域 F 上的非奇异 n 阶方阵 A 与任意 n 维向量 b, 本算法计算线性方程组 $Ax = b$ 的解 $x = A^{-1}b$.

1. 计算线性递推列 $(A^i b)_{i\in\mathbb{N}}$ 的极小多项式 $m \in F[X]$.
2. 返回 $x := -m_0^{-1} \sum_{1\le j\le d} m_j A^{j-1} b$.

以上给出了 Wiedemann 算法的整体过程, 但还有一个重要问题没有解决, 即算法第一步中要求的 n 维向量组成的线性递推列的极小多项式的计算. 由例 6.4 中 3, 我们知道 $a_b \equiv (A^j b)_{j\in\mathbb{N}}$ 与 $a_{ub} \equiv (u^T A^j b)_{\mathbb{N}}, u \in F^{\mathbb{N}}$ 的特征多项式有很重要的联系. 详言之, 我们知道 a_b 的每个特征多项式都将 a_{ub} 零化, 即 $\mathrm{Ann}(a_b) \subseteq \mathrm{Ann}(a_{ub})$, 从而 $m_{ub}|m_b$;另一方面, 若 $f \in \mathrm{Ann}(a_{ub}), \forall u \in F^{\mathbb{N}}$, 则 $f \in \mathrm{Ann}(a_b)$, 否则总可取 $u = f \bullet a_b$ 使得 $f \bullet a_{ub} \neq 0$. 这样, 我们有如下的随机性算法, 若该算法顺利执行完毕, 则能给出正确的结果.

算法6.10.

给定域 F 上 n 阶方阵 A 与 n 维向量 b, 该算法计算线性递推序列 $a_b \equiv (A^j b)_{j\in\mathbb{N}}$ 的极小多项式 m_b.

1. 随机选取 $u \in F^n$, 计算 $(u^T A^i b)_{i\in\mathbb{N}} \to a$.
2. 运用算法 6.8 计算 $m := m_{ub} \in F[X]$.
3. 校验步骤:若 $m(A)b \neq 0$, 进行步骤 1.
4. 返回 m.

注106. 如果计算结果在校验步骤中失败, 意味着 $m_{ub}|m_b$ 但 $m_{ub} \neq m_b$. 在以上算法中我们采取的是丢弃策略. 事实上, 注意到 m_{ub} 提供了 m_b 的一个非平凡因子. 因此我们也可以尝试如下两种改进方案, 将第 3 步的校验步骤扩展为以下两种方式中的任意一种:

- $m := m \cdot m_{ub}$, 若 $m(A)b \neq 0$, 则令 $b := m(A)b$, 返回步骤 1.
- $m := \mathrm{lcm}(m, m_{ub})$, 若 $m(A)b \neq 0$, 则返回步骤 1. [184]

以上两种做法的正确性都很容易证明.

我们还需要知道以上算法到底以多大概率得到正确结果. 为此, 我们需要对 $F^{\mathbb{N}}$ 作为 $F[X]$ 上模的结构有更清楚的认识.

设 $f \in F[X]$ 为 d 次多项式, 记 $\langle f\rangle = F[X]\cdot f$ 为由 f 生成的多项式理想. 定义 $M_f := \{a \in F^{\mathbb{N}} | f \bullet a = 0\}$, 则 M_f 为 $F[X]$ 上的循环模, 且 $M_f = F[X]\bullet c$, 其中 $c = (0, \cdots, 0, 1, c_d, c_{d+1}, \cdots)$, $c_d, c_{d+1}, \cdots$ 为根据前 d 个元素递推得到的序列元素. 这时 $c, x\bullet c, \cdots, x^{d-1}\bullet c$ 构成了 M_f 的一组基. 我们看到, $F[X]$ 在 M_f 上的作用给出了 $F[X]$ 到 M_f 的满同态, 从而决定了如下的模同构:

$$M_f \cong F[X]/\mathrm{Ann}(M_f) = F[X]/\langle f\rangle.$$

这样的结构给出了如下的引理, 它是我们分析算法有效性的基础.

引理6.3. *设 $A \in F^{n\times n}$, $b \in F^n\backslash 0$. $f \in F[X]$ 为 $(A^i b)_{i\in\mathbb{N}}$ 的极小多项式. 存在 $F-$ 线性的满射 $\psi : F^n \to F[X]/\langle f\rangle$, 使得 $\forall u \in F^n$, 以下两个命题等价:*

- *f 为 $(u^T A^i b)_{i\in\mathbb{N}} \in F^{\mathbb{N}}$ 的极小多项式.*
- *$\psi(u)$ 为单位, 即 $(\psi(u), f) = 1$.*

证明. 由前面的讨论, 存在 $F[X]/\langle f\rangle$ 到 M_f 的同构 $\phi : F[X]/\langle f\rangle \to M_f$. 我们还可定义 $F-$ 线性映射

$$\psi^* : F^n \ni u \longmapsto \psi^*(u) = (u^T A^i b)_{i\in\mathbb{N}} \in M_f$$

, 并由 f 为 M_f 的极小多项式得到 ψ^* 为满射. 因此可以构造 $F-$ 线性满映射

$$\psi = \phi^{-1}\circ\psi^* : F^n \to M_f,$$

用交换图表示如下:

$$\begin{array}{ccc} F[X]/\langle f\rangle & \xleftarrow{\psi=\phi^{-1}\circ\psi^*} & F^n \\ & {\scriptstyle\phi}\searrow \quad \swarrow{\scriptstyle\psi^*} & \\ & M_f & \end{array}.$$

这时, 我们有

$$\begin{aligned} f = m_{\psi^*(u)} &\Longleftrightarrow \forall g \in F[X](g\bullet\psi^*(u) = 0 \Leftrightarrow f|g) \\ &\Longleftrightarrow \forall g \in F[X]((g \bmod f)\bullet\psi(u) = 0 \Leftrightarrow g \bmod f = 0) \\ &\Longleftrightarrow \forall h \in F[X]/\langle f\rangle(h\bullet\psi(u) = 0 \leftrightarrow h = 0) \\ &\Longleftrightarrow \psi(u)\text{为单位.} \end{aligned}$$

□

定理6.15. 设 $U \subseteq F$, $A \in F^{n\times n}$, $b \in F^n\backslash 0$, f 为 $(A^ib)_{i\in\mathbb{N}}$ 的极小多项式, $d = \deg f$, 则 u 从 U^n 中随机选取时, f 为 $(u^TA^ib)_{i\in\mathbb{N}}$ 的极小多项式的概率 $p \geq 1 - d/\#U$.

这一定理给出了算法有效性的估计, 并可据此给出 Wiedemann 算法的平均复杂度估计.

证明. 记

$$u = (u_1, \cdots, u_n)^T = u_1e_1 + \cdots + u_ne_n \in F^n$$

为 F^n 中的任意向量, 由 ψ 线性得到

$$\begin{aligned}\psi(u) &= u_1\psi(e_1) + \cdots + u_n\psi(e_n)\\ &= (u_1h_1 + \cdots + u_nh_n) \bmod f,\end{aligned}$$

其中 $h_1, \cdots, h_n \in F[X]$, 且次数低于 d. $\psi(e_j) = h_j \bmod h_j$. 令 $y_1, \cdots, y_n$ 为不定元, $r = \mathrm{res}_x(y_1h_1 + \cdots + y_nh_n, f) \in F[y_1, \cdots, y_n]$, 则 r 的总次数不超过 d. 由于 $\psi(u)$单位 $\Leftrightarrow r(u_1, \cdots, u_n) \neq 0$, 但 ψ 满, 故 $r \neq 0$. 从而 $u_i \in U$ 随机选择时, $p \geq 1 - d/\#U$. □

根据该定理, 可以给出 Wiedemann 算法 6.9 的平均复杂度估计, 证明可参考 [174].

定理6.16. 若 F 中含有不少于 $2n$ 个元素, 则算法 *6.9* 的复杂度为 $4nc(A) + O(n^2)$, 其中 $c(A)$ 为矩阵 A 左乘向量所需的计算量, 与矩阵 A 的结构有关.

以上我们给出了有限域上适用于稀疏非奇异矩阵的 Wiedemann 算法及其有效性分析. 在此之后, 黑箱算法成为精确线性代数研究领域发展的主流. 首先是将这类算法应用于更广泛的情形, 如奇异系数矩阵的方程组, 元素很少的系数域上的方程组等, 并被用于解决线性代数中相关联的其他问题, 如矩阵的秩, 行列式等. 可参考 [80]2.3 节和 [171] 中包含的有关参考文献.

对于系数矩阵奇异情形, Wiedemann 本人在 [184] 中给出了一种基于随机蝶形变换(Benes 变换)的预处理步骤, 在此之后其他形式的预处理步骤以及相当于数值算法中的 Lanczos 迭代, 共轭梯度法等迭代方法的黑箱算法得到广泛的应用, 并在此基础上发展了块形式的相应算法, 如 [71][96][58][95] 等. 以上算法大多已经在LinBox[8]系统中得到实现 [70].

对于系数域元素很少的情形, 在同样预处理步骤下以上的有效性分析将失效. 这时, 一种做法是将系数域进行适当的扩张(这在 [184] 中有了初步的描述), 保证

[8]http://www.linalg.org

以上分析成立. 但这样做的代价是域扩张带来的复杂计算步骤. 在 LinBox 系统中, 并未采用域扩张的方法, 而是在前述方法上增加适当的校验步骤, 保证计算结果以较高的概率正确 [70].

7 一元多项式求值和插值

本章主要介绍一元多项式求值和插值的快速算法 [174], 它们本身都不是很复杂, 因此不打算用太多的篇幅介绍. 关于这一方面, 有兴趣的读者也可以参阅 [72], [88].

由于我们从本章开始将进入多项式的相关算法, 首先对一些通用的多项式记号作一简单说明. 设有多项式

$$f = \sum_{i=0}^{n} a_n x^n = a_0 + a_1 x + \cdots + a_n x^n,$$

则称 n 为 f 的次数, 记为 $\deg f$, 称 a_n 为 f 的领项系数(Leading Coefficient), 记为 $\mathrm{lc}(f)$, 称 x^n 为 f 的领项单项式(Leading Monomial), 记为 $\mathrm{lm}(f)$, 称 $a_n x^n$ 为 f 的领项(Leading Term), 记为 $\mathrm{lt}(f)$, 后文中也会提到首项, 即领项. 定义 f 的首一化多项式为

$$\mathrm{monic}(f) = \frac{f}{\mathrm{lc}(f)}.$$

7.1 求值算法

提起一元多项式的求值算法, 我们都会想起最著名的 Horner 规则, 即对于 $f(x) = \sum_{0 \le i < n} f_i x^i$, 利用下式来求其值:

$$f(x) = (\cdots(f_{n-1}x + f_{n-2})x + \cdots + f_1)x + f_0.$$

在该算法中, 需要计算总共 $n-1$ 次乘法和 $n-1$ 次加法. 如果要同时计算多项式在 n 个不同点处的值, 则需要 $O(n^2)$ 的计算量.

对于一般多项式的单点求值问题来说, Horner 规则已经是最优的了. Pan[135] 于 1966 年证明了 Horner 规则使用乘法的次数是最少的, 即 n 次多项式的求值至少要 n 次乘法.

但是对于多点求值来说, Horner 规则就不一定是最优的了. 下面给出一种快速多点求值算法, 其计算复杂度为 $O(M(n)\log n)$, 其中 $M(n)$ 表示 n 次多项式乘法计算的复杂度(参考 [174] 封三说明). 为了说明算法, 我们取 $n=2^k$ 是 2 的一整数次幂, 需要求值的点为 $u_0,u_1,\ldots,u_{n-1}$, 并且令 $m_j=x-u_j(j=0,\ldots,n-1)$. 下面构造一棵完全二叉树, 以 $M_{i,j}$ 表示从叶($i=0$)往上数第 i 层, 从该层左往右数第 j 个结点, 结点值为

$$M_{i,j}=\prod_{j\cdot 2^i\le l<(j+1)\cdot 2^i} m_l,$$

图7.1表示其结构.

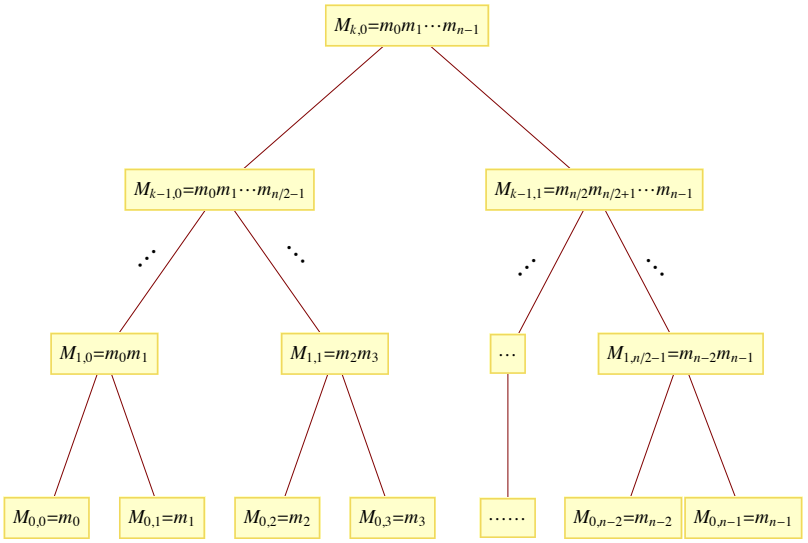

图 7.1: $M_{i,j}$ 二叉树示意图

其中每个叶结点都是一次式 $M_{0,j}=m_j$. 算法 7.1 给出了上面树的构造算法.

算法7.1 ($M_{i,j}$ 二叉树构造算法).

1. 对 $0\le j<n$, 令 $M_{0,j}=m_j$,
2. 对 i 从 1 循环到 k, 做下面第 3 步,

3. 对 j 从 0 循环到 2^{k-i}, $M_{i,j} = M_{i-1,2j}M_{i-1,2j+1}$.

利用上面的构造的二叉树, 我们可以实现快速求值算法.

算法7.2 (快速求值算法).

输入:多项式 f 和 n(满足 $\deg f < n$)个点 $u_0, \ldots, u_{n-1}$,

输出:$f(u_0), \ldots, f(u_{n-1})$.

1. 若 $n = 1$ 则输出 f(此时 f 为常数),
2. $r_0 = f \bmod M_{k-1,0}$, $r_1 = f \bmod M_{k-1,1}$,
3. 递归调用本算法求 $r_0(u_0), \ldots, r_0(u_{n/2-1})$,
4. 递归调用本算法求 $r_1(u_{n/2}), \ldots, r_1(u_{n-1})$,
5. 输出上面两步求出的 n 个值.

我们举例来说明上述算法.

例7.1. 求多项式 $f(x) = x^3 + 2x^2 + 3x + 4$ 在 $\{u_0, u_1, u_2, u_3\} = \{1, 2, 3, 4\}$ 四个点处的值.

解: 首先由 $f \bmod M_{1,0} = f \bmod (x-1)(x-2) = 16x - 6$ 和 $f \bmod M_{1,1} = f \bmod (x-3)(x-4) = 54x - 104$ 可以将问题化为求 $16x - 6$ 在 $1, 2$ 处的值和 $54x - 104$ 在 $3, 4$ 处的值. 再做一次模运算(模一次多项式实际上相当于求值)可以求出四个点处的值分别为 $10, 26, 58, 112$. ◇

注107. 以上讨论的都是 n 为 2 的整数次幂的情况, 而一般情况下这是不能满足的, 这时, 我们可以采取添一些项, 例如在多点求值时添一些不同的求值点以凑成 2 的整数次幂情形, 或者不必得到完全二叉树, 在递归建立 $M_{i,j}$ 的树或求值时每次将待处理的点分成数目大致相同的两组进行处理. 下一小节的插值算法当 n 不是 2 的整数次幂时, 也可同样处理.

7.2 插值算法

多项式的插值是指已知多项式在 n 个不同点处的值, 求出此多项式, 当然是在模 x^n 的意义下, 即次数不超过 n 时, 此多项式是唯一的. 首先我们想到了著名

的 Lagrange 插值算法, 设已知 n 个点 $u_0,\dots,u_{n-1}$ 和 n 次多项式 f 在这些点上的值 $v_0,\dots,v_{n-1}$, 记 $m_j=x-u_j$, $m=\prod_{0\le j<n} m_j$, $s_i=\prod_{j\ne i,0\le j<n}\frac{1}{u_i-u_j}$, 则 Lagrange 插值的结果可以表示为:

$$f=\sum_{0\le i<n}\frac{v_i s_i m}{m_i}.$$

这是一个复杂度 $O(n^2)$ 的算法(见 [174] 定理 5.1). 一元插值的方法还有 Newton 插值法, 其复杂度也是 $O(n^2)$ 的 [191], 见多元多项式插值部分的算法 11.1 .

下面提出的快速插值算法, 其复杂度与求值算法一样, 也为 $O(M(n)\log n)$. 为求插值多项式, 首先我们要求出 s_i, 因为

$$m'(u_i)=\sum_{0\le j<n}\frac{m}{m_j}\Big|_{x=u_i}=\frac{m}{m_i}\Big|_{x=u_i}=\frac{1}{s_i},$$

故 $s_i=\dfrac{1}{m'(u_i)}$. 现在令 $c_i=v_i s_i$, 并设 $n=2^k$ 是 2 的整数次幂.

算法7.3 (快速插值算法).

输入:$u_0,\dots,u_{n-1}$, $c_0,\dots,c_{n-1}$,

输出: $\sum_{0\le i<n}\frac{c_i m}{m_i}$.

1. 若 $n=1$ 则输出 c_0,
2. 递归调用本算法, 输入前 $n/2$ 个点, 求出 r_0,
3. 递归调用本算法, 输入后 $n/2$ 个点, 求出 r_1,
4. 输出 $M_{k-1,1}r_0+M_{k-1,0}r_1$.

算法有效性. 我们只需证明算法能返回结果 $\sum_{0\le i<n} c_i\prod_{0\le j<n,j\ne i} m_j$. 采用递归论证的方法, 我们假设在每一次递归调用执行下一级算法时, 均会得到正确的结果, 即

$$r_0=\sum_{0\le i<n/2} c_i\prod_{0\le j<n/2,j\ne i} m_j,\quad r_1=\sum_{n/2\le i<n} c_i\prod_{n/2\le j<n,j\ne i} m_j.$$

那么该步算法将返回结果:

$$M_{k-1,1}r_0+M_{k-1,0}r_1=r_0\prod_{n/2\le j<n} m_j+r_1\prod_{0\le j<n/2} m_j=\sum_{0\le i<n} c_i\prod_{0\le j<n,j\ne i} m_j,$$

即若下一级递归结果正确, 则该步算法也将返回正确结果. 另外, 在算法递归终止处, 即 $n=1$ 情况下, 算法返回 c_0, 此结果显然是正确的. 算法有效性得证. □

同样地, 我们给出一个具体的例子来说明本算法.

例7.2. 考虑 $\{u_0,u_1,u_2,u_3\}=\{1,2,3,4\}$, $\{v_0,v_1,v_2,v_3\}=\{10,26,58,112\}$, 求次数小于 4 的多项式 f 使得 $f(u_i)=v_i(1\leq i\leq 4)$.

解: 首先 $m=\prod_{0\leq i<4}(x-u_i)=x^4-10x^3+35x^2-50x+24$, 则 $m'=4x^3-30x^2+70x-50$. 于是

$$s_0^{-1}=m'(1)=-6, s_1^{-1}=m'(2)=2, s_2^{-1}=m'(3)=-2, s_3^{-1}=m'(4)=6.$$

由 $c_i=v_is_i$ 可得

$$c_0=-\frac{5}{3}, c_1=13, c_2=-29, c_3=\frac{56}{3}.$$

由于 $n=4$, 情况比较简单, 只需递归两次, 很容易得到第一级算法中的

$$r_0=-\frac{5}{3}(x-2)+13(x-1)=\frac{34}{3}x-\frac{29}{3}, r_1=-29(x-4)+\frac{56}{3}(x-3)=-\frac{31}{3}x+60,$$

则最后插值结果为:

$$(x-3)(x-4)r_0+(x-1)(x-2)r_1=x^3+2x^2+3x+4.$$

可以看到, 此结果与例 7.1 是相符合的. ◇

8 一元多项式的最大公因子

设 D 为 UFD(即唯一析因整环, 例如 $\mathbb{Z}$), F 为域(例如 $\mathbb{R}$), 则 $D[x], F[x]$ 也是 UFD, 对其上两多项式可讨论最大公因子(GCD)问题. 最古典的方法即是 Euclid 算法. 然而, 在上世纪六十年代末一些试验中(参见 [174] 注记6.1)发现这种算法在 $D[x]$ 或 $F[x]$ 中的系数增长很快, 甚至于指数阶增长. 本书开头即展示了这样的"中间表示膨胀"的例子.

为了解决计算过程中系数膨胀的问题, Collins, Brown 发现了 UFD 上的多项式模最大公因子算法, 并首先由 Brown[46] 于 1971 年提出. 对于多元情形, Mozes 和 Yun[130] 于 1973 年提出了基于 Hensel 提升方法的模因子算法. 下面对经典的 Euclid 算法和改进算法进行论述.

本章假设读者具有一定的代数知识, 对 UFD, PID(主理想整环), 多项式及其 GCD 以及 Euclid 算法等有一定的了解. 关于基本的代数知识, 可参考高等代数学方面的书, 如 [12].

8.1 Euclid 算法

在 Euclid 整环 $F[x]$ 中, 我们有如下的 Euclid 除法:

定义8.1 (Euclid 除法)**.** 设 $f, g \in F[x]$, 其中 $g \neq 0$, 则存在唯一的 $q, r \in F[x]$ 使得

$$f = qg + r,$$

其中 $\deg r < \deg g$. 我们称此除法过程为 Euclid 除法, 并以 $f \operatorname{quo} g$ 记商 q, $f \operatorname{rem} g$ 记余式 r.

在一般高等代数教材(如 [12])中都证明了如下定理.

定理8.1. *在 Euclid 整环 $F[x]$ 中, 若 $h=\gcd(f,g)$, 则有唯一的非平凡多项式 s,t 使得*

$$sf+tg=h$$

且 $\deg s<\deg g, \deg t<\deg f$. 等式 $sf+tg=h$ 称为 Bezout 等式, s,t 称为 Bezout 系数.

对于 Euclid 整环 $F[x]$, 我们熟知有以下的扩展 Euclid 算法 8.1 . 注意到将算法中的 s,t 等计算舍去即为普通的 Euclid 算法.

算法8.1 (扩展 Euclid 算法).

输入: $F[x]$ 上多项式 f, g,

输出: f, g 的最大公因子 u, Bezout 等式中的系数 $S=(s,t)$ 使得 $sf+tg=u$.

1. $u=f, v=g, S=(1,0), T=(0,1)$,
2. 如果 $v=0$, 则转到第 4 步,
3. $r=u \operatorname{rem} v, q=u \operatorname{quo} v, L=S-qT, u=v, v=r, S=T, T=L$, 并转回第 2 步,
4. 输出 u, S.

由于我们要讨论的是"精确"的符号计算, 所以多项式的系数一般为整数或有理数. 而对于 $\mathbb{Q}[x]$ 中的多项式, 我们总可以找到整数乘以该多项式使其化为 $\mathbb{Z}[x]$ 中的问题. 因此, 后面关于多项式的符号运算集中讨论 $\mathbb{Z}[x]$ 中的情形.

为了引出 $\mathbb{Z}[x]$ 中求 GCD 的算法, 我们先引进如下一些定义和定理.

定义8.2 (本原多项式). 容度$\operatorname{cont}(f)$ 定义为:

$$\operatorname{cont}(f)=\operatorname{sgn}(a_n)\gcd(a_0,a_1,\ldots,a_n),$$

其中 $f=\sum a_i x^i \in \mathbb{Z}[x]$, 且 gcd 对单项的定义为

$$\gcd(a_0)=a_0,$$

f 的本原部分(primitive part)定义为 $\operatorname{pp}(f)=f/\operatorname{cont}(f)$, 若 $f=\operatorname{pp}(f)$, 则称其为本原多项式.

注108. 定义中关于多个整数的 GCD 是指取了绝对值之后的最大公因子, 之所以在容度的定义中乘上 $\operatorname{sgn}(a_n)$ 一项, 是为了使得其后定义的本原多项式的首项系数为正数.

命题8.1. 设 $f \in \mathbb{Z}[x], c \in \mathbb{Z}$, 我们有 $\operatorname{cont}(cf) = \operatorname{cont}(c)\operatorname{cont}(f), \operatorname{pp}(cf) = \operatorname{pp}(c)\operatorname{pp}(f)$.

定理8.2 (Gauss 引理). 设 $f, g \in \mathbb{Z}[x]$, 若 f, g 本原, 则 $h = fg$ 也是本原的.

类似还有以下一些命题, 相关内容可参阅相关的高等代数学内容(例如 [6]).

命题8.2. 设 $f, g \in \mathbb{Z}[x], \operatorname{cont}(fg) = \operatorname{cont}(f)\operatorname{cont}(g), \operatorname{pp}(fg) = \operatorname{pp}(f)\operatorname{pp}(g)$.

命题8.3. 设 $f, g \in \mathbb{Z}[x], h = \gcd(f, g) \in \mathbb{Z}[x]$, 则

$$\operatorname{cont}(h) = \gcd(\operatorname{cont}(f), \operatorname{cont}(g)) \in \mathbb{Z},$$

$$\operatorname{pp}(h) = \gcd(\operatorname{pp}(f), \operatorname{pp}(g)) \in \mathbb{Z}[x].$$

于是有下面的:

算法8.2 (一个整系数多项式的 Euclid 算法).
输入: $\mathbb{Z}[x]$ 上的多项式 f, g,
输出: f, g 在 $\mathbb{Z}[x]$ 上的最大公因子 h.

1. $h = \gcd(f, g) \in \mathbb{Q}[x]$,
2. $b = \gcd(\operatorname{lc}(f), \operatorname{lc}(g))$,
3. 输出 bh.

尽管 $\mathbb{Z}[x]$ 不是 Euclid 整环, 但我们可引入 $\mathbb{Z}[x]$ 上的伪除法, 来构造类似 Euclid 余式序列, 以此来求总大公因子. 当然这里的 $\mathbb{Z}$ 换为一般的 UFD 也是可以的.

定义8.3 (伪除法). 设 $f, g \in \mathbb{Z}[x]$, 若有 $\alpha, \beta \in \mathbb{Z}$, $q, r \in \mathbb{Z}[x]$ 使得

$$\alpha f = qg + \beta r,$$

其中 $\deg r < \deg g$, r 是本原多项式, 我们称此过程为多项式的伪除法, 并以 $f \operatorname{pquo} g$ 记伪商 q, $f \operatorname{prem} g$ 记伪余式 r.

定义8.4 (多项式余式序列). 若 $f,g\in\mathbb{Z}[x]$, 且 $\deg(f)\geq\deg(g)$, 令 $R_0=f,R_1=g$, 依次作伪除法 $\alpha_iR_{i-1}=Q_iR_i+\beta_iR_{i+1}$, $i=1,2,\ldots,k$, 满足 $R_{k-1}\operatorname{prem}R_k=0$, 则 $R_0,R_1,\ldots,R_k$ 称为 f,g 的多项式余式序列.

使用伪除法求最大公因子时就会导致余式序列的系数增长很快, 我们可以用下面定义的几种多项式余式序列一定程度上减小系数增长速度(参见 [13]). 本章稍后介绍的素数模方法能够有效抑制系数膨胀效应.

定义8.5 (几种多项式余式序列). 若记 $\delta_i=\deg(R_{i-1})-\deg(R_i)$, 则有如下定义:

(1)通常的伪余式序列: $\alpha_i=(\operatorname{lc}(R_i))^{\delta_i+1},\beta_i=\operatorname{cont}(R_{i-1}\operatorname{prem}R_i)$.

(2)子结式余式序列:

$$\alpha_i=(\operatorname{lc}(R_i))^{\delta_i+1},\beta_1=(-1)^{\delta_1+1},$$

$$\beta_i=-(\operatorname{lc}(R_{i-1}))\psi_i^{\delta_i},(2\leq i\leq k),$$

$$\psi_1=-1,\psi_i=(-\operatorname{lc}(R_{i-1}))^{\delta_{i-1}}\psi_{i-1}^{1-\delta_{i-1}},(2\leq i\leq k),$$

此方法计算量最小.

(3)约化多项式余式序列:

$$\alpha_i=(\operatorname{lc}(R_{i-1}))^{\delta_i+1},$$

$$\beta_1=1,\beta_i=\alpha_{i-1},(2\leq i\leq k).$$

8.2 域上多项式的快速 Euclid 算法

1938 年 Lehmer 最先提出了快速 Euclid 算法[114], 后来 Knuth[103], Schönhage[155], Moenck[124], Schwartz[156], Strassen[166] 对这些算法也有论述.

在域 F(例如 $\mathbb{F}_p$)上的多项式环中, Euclid 算法可表示为(其中各 r_i 均为首一的):

$$r_0=f,\quad r_1=g,\quad s_0=t_1=1,\quad s_1=t_0=0,$$

及

$$\begin{array}{lll}\rho_2r_2=r_0-q_1r_1, & \rho_2s_2=s_0-q_1s_1, & \rho_2t_2=t_0-q_1t_1,\\ \vdots & \vdots & \vdots\\ 0=r_{l-1}-q_lr_l, & \rho_{l+1}s_{l+1}=s_{l-1}-q_ls_l, & \rho_{l+1}t_{l+1}=t_{l-1}-q_lt_l,\end{array}\tag{8.1}$$

若设 $\rho_{l+1}=1, r_{l+1}=0$, 记

$$Q_i=\begin{pmatrix}0 & 1\\ \rho_{i+1}^{-1} & -q_i\rho_{i+1}^{-1}\end{pmatrix},$$

则有

$$\begin{pmatrix}r_i\\ r_{i+1}\end{pmatrix}=Q_i\begin{pmatrix}r_{i-1}\\ r_i\end{pmatrix}.$$

记 $R_i=Q_iQ_{i-1}\cdots Q_1$, 则有

$$R_i=\begin{pmatrix}s_i & t_i\\ s_{i+1} & t_{i+1}\end{pmatrix},\quad \begin{pmatrix}r_i\\ r_{i+1}\end{pmatrix}=R_i\begin{pmatrix}r_0\\ r_1\end{pmatrix}.$$

在随机情况下, 可证明域 $\mathbb{F}_p$ 中 $\deg r_2<\deg r_1-1$ 的概率 $\mathrm{P}(\deg r_2<\deg r_1-1)=\dfrac{1}{p}$, 当素数 p 很大时, 可认为 r_i 序列的下降速度很慢. 如果是有理数域上的多项式, 可以想到, 余式次数下降得应该更慢. 引入快速 Euclid 算法能在一定程度上补偿次数下降过慢导致的计算代价. 为了说明快速 Euclid 算法, 我们先引入下面一些定义和定理. 该算法的基本原理在于利用多项式的前若干项系数来计算余式序列.

以下我们简记多项式系数域为 F, 当然, 它可以是 $\mathbb{F}_p$ 或者 $\mathbb{Q}$.

定义8.6 (截式)**.** 对于 $f=\sum_{i=0}^{n}f_ix^i\in F[x], k\in\mathbb{Z}$, k-截式(k-truncated polynomial)定义为

$$f\upharpoonright k=f_nx^k+f_{n-1}x^{k-1}+\cdots+f_{n-k},$$

当 $i<0$ 我们约定 $f_i=0$. 因此当 $k\geq 0$ 时, $f\upharpoonright k$ 即取 f 的前 $k+1$ 个系数作一个新的 k 次多项式, 而当 $k<0$ 时 $f\upharpoonright k=0$.

注109. 当 $k\leq n$ 时, 有 $f\upharpoonright k=f\operatorname{quo}x^{n-k}$, 而当 $k>n$ 时有 $f\upharpoonright k=fx^{k-n}$.

注110. $\forall i\geq 0, (fx^i)\upharpoonright k=f\upharpoonright k$.

定义8.7 (k-度重合)**.** 设非零多项式 $f,g,f^*,g^*\in F[x]$, 且 $\deg f\geq\deg g, \deg f^*\geq\deg g^*, k\in\mathbb{Z}$, 则称 (f,g) 与 (f^*,g^*) k-度重合(coincide up to k), 如果

$$f\upharpoonright k=f^*\upharpoonright k$$

$$g\upharpoonright(k-\deg f+\deg g)=g^*\upharpoonright(k-\deg f^*+\deg g^*),\tag{8.2}$$

此时记为 $(f,g)\overset{k}{\sim}(f^*,g^*)$, 可验证 $\sim$ 为一等价关系.

根据式 (8.2) , 此等价关系有如下性质:

命题8.4. *若* $(f,g) \overset{k}{\sim} (f^*,g^*)$ *且* $k \geq \deg f - \deg g$, *则* $\deg f - \deg g = \deg f^* - \deg g^*$.

关于 k-度重合, 有如下重要的命题:

定理8.3. *对于* $k \in \mathbb{Z}$, *若非零多项式* f,g,f^*,g^* *满足* $(f,g) \overset{2k}{\sim} (f^*,g^*)$, *且* $k \geq \deg f - \deg g \geq 0$, *若有 Euclid 除法*

$$f = qg + r, \quad f^* = q^*g^* + r^*,$$

则

1. $q = q^*$,
2. $(g,r) \overset{2(k-\deg q)}{\sim} (g^*,r^*)$ *或者* $r = 0$ *或者* $k - \deg q < \deg g - \deg r$.

证明. 对 (f,g) 和 (f^*,g^*) 乘以 x 的适当幂次后, 可使 $\deg f = \deg f^* > 2k$ 成立, 不妨设命题中的多项式已满足此式, 则由命题 8.4 有 $\deg g = \deg g^*$ 及 $k \geq \deg q = \deg f - \deg g = \deg f^* - \deg g^* = \deg q^*$. 下面分两部分证明本定理.

(1)首先我们有如下三个不等式

$$\begin{aligned}
&\deg(f - f^*) < \deg f - 2k \leq \deg g - k(\text{注意}k\text{-截式取多项式的前}k+1\text{项}),\\
&\deg(g - g^*) < \deg g - (2k - (\deg f - \deg g))\\
&\qquad\qquad = \deg f - 2k \leq \deg g - k \leq \deg g - \deg q,\\
&\deg(r - r^*) \leq \max\{\deg r, \deg r^*\} < \deg g,
\end{aligned}$$

根据上面的不等式, 由 $f - f^* = q(g-g^*) + (q-q^*)g^* + (r-r^*)$ 可知 $\deg[(q-q^*)g^*] < \deg g$, 故 $q = q^*$.

(2)假定 $r \neq 0$ 且 $k - \deg q \geq \deg g - \deg r$, 则由 $(f,g) \overset{2k}{\sim} (f^*,g^*)$ 有

$$g \upharpoonright 2(k - \deg q) = g^* \upharpoonright 2(k - \deg q),$$

另外

$$\begin{aligned}
\deg(r - r^*) &\leq \max\{\deg(f - f^*), \deg q + \deg(g - g^*)\}\\
&< \deg q + \deg f - 2k\\
&= \deg g - 2(k - \deg q)\\
&= \deg r - [2(k - \deg q) - (\deg g - \deg r)],
\end{aligned}$$

又由假设有 $\deg r \geq \deg q + \deg g - k \geq \deg q + \deg f - 2k \Rightarrow \deg r = \deg r^*$, 于是 $r \upharpoonright [2(k - \deg q) - (\deg g - \deg r)] = r^* \upharpoonright [2(k - \deg q) - (\deg g^* - \deg r^*)]$, 故 $(g,r) \overset{2(k-\deg q)}{\sim} (g^*,r^*)$. □

下面考虑两个首一的多项式 Euclid 余式序列

$$\begin{aligned} r_0 &= q_1 r_1 + \rho_2 r_2 & r_0^* &= q_1^* r_1^* + \rho_2^* r_2^* \\ &\vdots & &\vdots \\ r_{l-1} &= q_l r_l & r_{l^*-1}^* &= q_{l^*}^* r_{l^*}^* \end{aligned}$$

分别令 $m_i = \deg q_i, m_i^* = \deg q_i^*, n_i = \deg r_i = n_0 - m_1 - \cdots - m_i$, 对 $k \in \mathbb{N}$, 定义 $\eta(k) = \max\{0 \le j \le l \mid \sum_{1\le i\le j} m_i \le k\}$, 则我们有

$$n_0 - n_{\eta(k)} = \sum_{1\le i\le \eta(k)} m_i \le k < \sum_{1\le i\le \eta(k)+1} m_i = n_0 - n_{\eta(k)+1}.$$

下面的定理是快速 Euclid 算法的基础:

定理8.4. *对于 $k \in \mathbb{N}, h = \eta(k), h^* = \eta^*(k)$, 若*

$$(r_0, r_1) \overset{2k}{\sim} (r_0^*, r_1^*),$$

则 $h = h^*$, $q_i = q_i^*, \rho_{i+1} = \rho_{i+1}^*$ 对 $1 \le i \le h$ 成立.

证明. 只需对 $0 \le j \le h$ 的 j 进行归纳, 证明如下命题: $j \le h^*, q_i = q_i^*, \rho_{i+1} = \rho_{i+1}^*$ 对于 $1 \le i \le j$ 成立, 且 $j = h$ 或

$$(r_j, r_{j+1}) \overset{s}{\sim} (r_j^*, r_{j+1}^*), \text{其中} s = 2(k - \sum_{1\le i\le j} m_i).$$

$j = 0$ 时命题无须论证, 由定理 8.3 可以证明归纳过程. 另外

$$\sum_{1\le i\le j} \deg q_i^* = \sum_{1\le i\le j} \deg q_i \le \sum_{1\le i\le h} \deg q_i \le k$$

可知 $j \le \eta^*(k) = h^*$, 证毕. □

由此我们可以得到下面的算法:

算法8.3 (快速扩展 Euclid 算法)**.**

输入: 首一多项式 $r_0, r_1 \in F[x], k \in \mathbb{N}(0 \le k \le n)$, 其中 $n = n_0 = \deg r_0 > n_1 = \deg r_1$,

输出: $h = \eta(k)$ 以及 $R_h \in M_{2\times 2}(F[x])$.

1. 如果 $r_1 = 0$ 或 $k < n_0 - n_1$, 则输出 $0, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$,

2. $d = \lfloor k/2 \rfloor$,
3. 递归调用本算法, 输入 $r_0 \vdash 2d, r_1 \vdash (2d - (n_0 - n_1)), d$, 并将结果输出至 $j - 1 = \eta(d), R = Q_{j-1} \cdots Q_1$,
4. 赋值 $\begin{pmatrix} r_{j-1} \\ r_j \end{pmatrix} = R \begin{pmatrix} r_0 \\ r_1 \end{pmatrix}$, $\begin{pmatrix} n_{j-1} \\ n_j \end{pmatrix} = \begin{pmatrix} \deg r_{j-1} \\ \deg r_j \end{pmatrix}$,
5. 如果 $r_j = 0$ 或者 $k < n_0 - n_j$ 则输出 $j - 1, R$,
6. 赋值 $q_j = r_{j-1} \operatorname{quo} r_j$, $\rho_{j+1} = \operatorname{lc}(r_{j-1} \operatorname{rem} r_j)$, $r_{j+1} = \operatorname{monic}(r_{j-1} \operatorname{rem} r_j)$, $n_{j+1} = \deg r_{j+1}$, $d^* = k - (n_0 - n_j)$,
7. 递归调用本算法, 输入 $r_j \vdash 2d^*, r_{j+1} \vdash (2d^* - (n_j - n_{j+1})), d^*$, 并将结果输出至 $h - j = \eta(d^*), S = Q_h \cdots Q_{j+1}$,
8. 赋值 $Q_j = \begin{pmatrix} 0 & 1 \\ \rho_{j+1}^{-1} & -q_j \rho_{j+1}^{-1} \end{pmatrix}$,
9. 输出 h, SQ_jR.

注111. 理论上每次取 k 时应当使恰好只计算一步, 这样可以达到最高的效率, 即既不多取了不必要的系数进行计算, 又不致于系数取得不够. 当余式序列的次数 n_i 下降得缓慢时, 相应的 k 就可以取得比较小. 实际中无法根据 n_i 的信息选择, 因而有上面的二分递归的策略来选择.

注112. 初始进行函数调用时, 我们可取 $k = n$, 根据 $\eta(k)$ 的定义我们知道 $\eta(k) = l$, 因而算法必将返回 l 和 R_l, 根据 R_l 可以得到 r_l, 即最大公因子.

本章稍后将会介绍 $\mathbb{Z}[x]$ 上的素数模公因子算法, 由于 $\mathbb{F}_p$ 为域, 可利用本节所提到的快速 Euclid 算法提高计算效率.

8.3 结式性质及其计算

8.3.1 结式

对于多项式最大公因子问题, 结式理论纯粹是一个概念上的工具, 并没有直接用于实际算法中. 但结式理论对后面的 GCD 算法理论有重要的作用. 另外结式的计算在某些问题中也十分重要, 如符号积分中的 Rothstein-Trager 结式等等. 本小

节先介绍结式的定义及一些相关性质, 后面将会介绍利用多项式余式序列(PRS)来计算它的方法.

引理8.1. *设 F 是域, 非零多项式 f, $g \in F[x]$. 那么 $\gcd(f,g) \neq 1$ 当且仅当 $\exists s,t \in F[x]\setminus\{0\}$ 使得 $sf+tg=0$, 且 $\deg s < \deg g$, $\deg t < \deg f$.*

证明. 令 $h=\gcd(f,g)$, 若 $f \neq 1$, 则 $\deg h \geq 1$, 令 $s=-g/h$, $t=f/h$ 则满足.

反过来, 设存在这样的 s, t, 若 f, g 互素, 则 $sf=-tg \Rightarrow f \mid t$, 这与 $t\neq 0$ 且 $\deg f > \deg t$ 矛盾, 因此 $h=1$. □

注113. 当 F 只是 UFD 时上面的引理仍成立, 不过引理的条件 $\gcd(f,g)\neq 1$ 应理解为 f,g 在该 UFD 分式域为系数的多项式环中的最大公因子, 或者说 $\gcd(f,g)$ 非平凡, 即非常数多项式.

为了引出结式的定义, 我们仍需给出如下一些定义及定理.

定义8.8. 对于 $d\in\mathbb{N}$, 定义 $P_d=\{a\in F[x] \mid \deg a < d\}$.

定义8.9. 对于给定的 n, m 次多项式 f, $g\in F[x]$, 定义

$$\varphi=\varphi_{f,g}: F[x]\times F[x]\to F[x], \qquad (s,t)\mapsto sf+tg,$$

并令 $\varphi_0: P_m\times P_n \to P_{n+m}$ 为 φ 在 $P_m\times P_n$ 上的限制.

定理8.5. *设 f, $g\in F[x]$ 为 n, m 次非零多项式, 那么有下面的结论:*

1. $\gcd(f,g)=1 \Leftrightarrow \varphi_0$ *是同构.*
2. *若* $\gcd(f,g)=1$, *则 f, g 的 Bezout 系数 s, t 构成 $\varphi_0(s,t)=1$ 的唯一解.*

证明. 由引理 8.1 知 $\gcd(f,g)=1 \Leftrightarrow \varphi_0$ 是单射.

因为对于相同维数线性空间之间的线性映射, 单射等价于同构, 所以第一条结论满足. 由 φ_0 为同构知这样的 s, t 也是唯一的. □

如果记 $(s,t)=(y_{m-1},\dots,y_0,z_{n-1},\dots,z_0)^T$, 其中 $s=\sum_{0\leq j<m} y_j x^j$, $t=\sum_{0\leq j<n} z_j x^j$, 则线性映射 φ_0 可在自然基 $\{(x^j,0)\}_{0\leq j\leq m-1}, \{(0,x^j)\}_{0\leq j\leq n-1}$ 下表示为如下矩阵 S, 称为 f, g 的 Sylvester 矩阵:

$$S=\begin{pmatrix}
f_n & & & & g_m & & & & & \\
f_{n-1} & f_n & & & g_{m-1} & g_m & & & & \\
\vdots & \vdots & \ddots & & \vdots & \vdots & \ddots & & & \\
\vdots & \vdots & & f_n & g_1 & \vdots & & \ddots & & \\
\vdots & \vdots & & f_{n-1} & g_0 & \vdots & & & \ddots & \\
\vdots & \vdots & & \vdots & & g_0 & & & & g_m \\
f_0 & \vdots & & \vdots & & & \ddots & & & \vdots \\
 & f_0 & & \vdots & & & & \ddots & & \vdots \\
 & & \ddots & \vdots & & & & & \ddots & \vdots \\
 & & & f_0 & & & & & & g_0
\end{pmatrix}.$$

上面的定理用矩阵语言描述即为:

推论8.1. 设 S 为 f, g 的 *Sylvester* 矩阵, 则有

1. $\gcd(f,g)=1 \Leftrightarrow \det S \neq 0$,
2. 若 $\gcd(f,g)=1$, 且 $y_0,y_1,\ldots,y_{m-1},z_0,\ldots,z_{n-1}\in F$ 满足

$$S(y_{m-1},\ldots,y_0,z_{n-1},\ldots,z_0)^T=(0,0,\ldots,0,1)^T,$$

那么 $s=\sum_{i=0}^{m-k}y_ix^i$, $t=\sum_{i=0}^{n-k}z_ix^i$ 为 f, g 的 *Bezout* 系数, 使得 $sf+tg=1$.

到此, 我们可以给出结式的定义了.

定义8.10. 设 R 为 UFD, $f,g\in R[x]$, $S=\mathrm{Syl}(f,g)$ 为它们的 Sylvester 矩阵, 我们称行列式 $\det S$ 为结式, 记作 $\mathrm{res}(f,g)$.

注114. 当 $n=m=0$ 时 S 是空矩阵, 我们约定结式为 1. 若 f 为 0 或非常数多项式, 我们约定 $\mathrm{res}(f,0)=\mathrm{res}(0,f)=0$, 若 f 是非零常数则约定 $\mathrm{res}(f,0)=\mathrm{res}(0,f)=1$. 前面定理在这些情形下仍然成立.

下面是一些关于结式的性质, 对于以后的算法分析是很有用的.

推论8.2. 设 F 为域, 非零多项式 $f,g\in F[x]$, 则下面各条件等价:

1. $\gcd(f,g)=1$,
2. $\mathrm{res}(f,g)=\deg S\neq 0$,

3. 不存在非零多项式 $s,t\in F[x]$ 使得

$$sf+tg=0,\quad \deg s<\deg g,\quad \deg t<\deg f.$$

推论8.3. 设 R 为 UFD, 非零多项式 $f,g\in R[x]$. 则 $\gcd(f,g)$ 非平凡当且仅当 $\operatorname{res}(f,g)=0$.

推论8.4. 若 R 为 UFD, 非零多项式 $f,g\in R[x]$, 则存在非零多项式 $s,t\in R[x]$ 使得 $sf+tg=\operatorname{res}(f,g)$ 且 $\deg s<\deg g,\deg t<\deg f$.

证明. 令 F 是 R 的分式域. 若 $r=\operatorname{res}(f,g)=0$, 则由推论 8.2 知存在满足要求的 $s,t\in F[x]$, 再将其乘一个公分母即可. 若 $r\neq 0$, 则 f,g 互素, 于是存在 $s^*,t^*\in F[x]$ 满足 $\deg s^*<\deg g,\deg t^*<\deg f$ 且 $s^*f+t^*g=1$, 由线性方程组的 Cramer 法则知 $s=rs^*,t=rt^*$ 满足所需要求. □

引理8.2. 设 R 为 UFD, 非零多项式 $f,g\in R[x]$, $r=\operatorname{res}(f,g)\in R$, $I\subset R$ 为一理想, 记 R 中元素 a 模 I 的像为 $\overline{a}$, 设 $\overline{\operatorname{lc}(f)}\neq 0$. 则

1. $\overline{r}=0\Leftrightarrow \operatorname{res}(\overline{f},\overline{g})=0$,

2. 若 R/I 是 UFD, 则 $\overline{r}=0\Leftrightarrow \gcd(\overline{f},\overline{g})$ 非平凡.

证明. 由 $\overline{\operatorname{lc}(f)}\neq 0$ 可知 $\operatorname{Syl}(\overline{f},\overline{g})$ 的"尺寸"并不会减小, 因此有 $\overline{r}=0\Leftrightarrow \operatorname{res}(\overline{f},\overline{g})=0$.

再假设 $\overline{g}\neq 0$, 令 i 为最小的指标使得 $\overline{g_{m-i}}\neq 0$, 此时将 Sylvester 矩阵分块, 去掉左 i 列和上 i 行, 只留下右下 $(m+n-i)\times(m+n-i)$ 的方阵 M, 显然 $M=\operatorname{Syl}(\overline{f},\overline{g})$, 于是 $\overline{r}=\overline{f_n^i\det M}=\overline{f_n^i}\operatorname{res}(\overline{f},\overline{g})$, 可知第一条成立. 再由推论 8.3 可知第二条成立. □

8.3.2 Euclid 算法计算结式

下面我们将要介绍用 Euclid 辗转相除算法计算结式的方法, 即多项式余式序列算法(PRS).

我们先对 Euclid 辗转相除求最大公因子的过程作一些细致的考虑. 设 F 为域, f, g 是 $F[x]$ 上非零多项式, $\deg f=n>m=\deg g$. 我们仍采用式 (8.1) 中的记号, 将 Euclid 算法的过程表示为

$$\rho_0 r_0=f,\quad \rho_1 r_1=g,\quad \rho_0 s_0=\rho_1 t_1=1,\quad \rho_1 s_1=\rho_0 t_0=0,$$

及

$$\begin{array}{lll} \rho_2 r_2 = r_0 - q_1 r_1, & \rho_2 s_2 = s_0 - q_1 s_1, & \rho_2 t_2 = t_0 - q_1 t_1, \\ \vdots & \vdots & \vdots \\ 0 = r_{l-1} - q_l r_l, & \rho_{l+1} s_{l+1} = s_{l-1} - q_l s_l, & \rho_{l+1} t_{l+1} = t_{l-1} - q_l t_l, \end{array}$$

注115. 注意此时各 r_i 均为首一的, 由于我们没有假设 f, g 是首一的, 因此 ρ_0, ρ_1 的出现是使得 f, g 首一化.

定义8.11. $n_i = \deg r_i (0 \leq i \leq l+1)$ 称为次数序列.

定理8.6. *设 $0 \leq k \leq m \leq n$, 则 k 不出现在次数序列当且仅当存在 $s, t \in F[x]$, 满足*

$$t \neq 0, \quad \deg s < m-k, \quad \deg t < n-k, \quad \deg(sf+tg) < k.$$

证明. 必要性:若 k 不在次数序列中, 则存在 $i(2 \leq i \leq l+1)$ 使得 $n_i < k < n_{i-1}$, 于是 $s_i f + t_i g = r_i, \deg r_i = n_i < k$, 且 $\deg s_i = m - n_{i-1} < m-k$, $\deg t_i = n - n_{i-1} < n-k$. 最后两个不等式可以用 Euclid 辗转相除的过程归纳证明. 这样的 s_i, t_i 即可当作满足定理条件的 s, t. 而当 $i = l+1$ 时, 情况稍有不同, 此时令 $s = g/r_l, t = -f/r_l, k < n_l$ 则 $sf + tg = r_{l+1} = 0$, 于是 $\deg r_{l+1} = -\infty < k$ 仍然满足定理条件.

充分性:由 [174] 引理 5.15 可知 $\exists i(1 \leq i \leq l+1)$ 和 $\alpha \in F[x] \setminus \{0\}$ 使得 $t = \alpha t_i,$, $s = \alpha s_i$, $r = sf + tg = \alpha r_i$, 于

$$\begin{aligned} n - n_{i-1} \leq \deg \alpha + n - n_{i-1} = \deg(\alpha t_i) = \deg t < n-k, \\ n_i \leq \deg \alpha + n_i = \deg(\alpha r_i) = \deg r < k, \end{aligned}$$

故 $n_i < k < n_{i-1}$. □

为了将上面的内容翻译成一种代数语言, 我们考虑上一节定义的线性映射 φ 在 $P_{m-k} \times P_{n-k}$ 上的限制. 容易知道, φ 的象是在空间 P_{n+m-k} 中, 为了使得其成为同构, 我们要使两个空间维数相等. 进而我们考虑了如下的线性映射:

$$\varphi_k : P_{m-k} \times P_{n-k} \to P_{n+m-2k}, \quad (s,t) \mapsto (sf+tg) \text{ quo } x^k.$$

我们有如下的推论:

推论8.5. *记号同前, 我们有:*

1. *k 在次数序列 $\{n_i\}$ 中当且仅当 φ_k 是同构.*

2. 若 $k=n_i$, 则 s_i,t_i 是 $\varphi_k(s_i,t_i)=1$ 的唯一解.

我们记 φ_k 的在自然基下的矩阵为 S_k, 则 S_k 为 Sylvester 矩阵的子矩阵:

$$\begin{pmatrix}
f_n & & & & g_m & & & \\
f_{n-1} & f_n & & & g_{m-1} & g_m & & \\
\vdots & & \ddots & & \vdots & & \ddots & \\
f_{n-m+k+1} & \cdots & \cdots & f_n & g_{k+1} & \cdots & \cdots & g_m \\
\vdots & & & \vdots & \vdots & & & \vdots \\
f_{k+1} & \cdots & \cdots & f_m & g_{m-n+k+1} & & & \vdots \\
\vdots & & & \vdots & \vdots & & & \vdots \\
f_{2k-m+1} & \cdots & \cdots & f_k & g_{2k-n+1} & \cdots & \cdots & g_k
\end{pmatrix}. \tag{8.3}$$

推论8.6. 记号同前, 我们有

1. k 在次数序列中当且仅当 $\det S_k\neq 0$.

2. 若 $k=n_i$, 且 $y_0,\dots,y_{m-k-1},z_0,\dots,z_{n-k-1}\in F$ 是下面的线性方程

$$S_k(y_{m-k-1},\dots,y_0,z_{n-k-1},\dots,z_0)^T=(0,\dots,0,1)^T$$

的唯一解, 则 $s_i=\sum y_jx^j$, $t_i=\sum z_jx^j$.

定义8.12. 记 $\sigma_k=\det S_k$.

现在我们来逐步推导 σ_k 的计算方法, 注意到求出 σ_0 即是求出了结式. 首先我们有如下引理:

引理8.3. 设 $f,g,r\in F[x]$ 首一且次数分别为 n,m,d, 其中 $n\geq m$, 且

$$\rho r=f\operatorname{rem} g,\rho\in F\setminus\{0\},$$

则对于 $k(0\leq k\leq d)$ 有

$$\sigma_k(f,g)=(-1)^{(n-k)(m-k)}\rho^{m-k}\sigma_k(g,r).$$

证明. 设 $f=qg+\rho r$, q 为 $(n-m)$ 次多项式, 于是

$$\begin{pmatrix}1\\ f_{n-1}\\ \vdots\\ f_0\end{pmatrix}-\begin{pmatrix}1 & & \\ g_{m-1} & \ddots & \\ \vdots & & 1\\ g_0 & & \vdots\\ & \ddots & \vdots\\ & & g_0\end{pmatrix}\begin{pmatrix}q_{n-m}\\ \vdots\\ q_0\end{pmatrix}=\begin{pmatrix}0\\ \vdots\\ 0\\ \rho r_d\\ \vdots\\ \rho r_0\end{pmatrix}.$$

对 $\det S_k$ 作初等列变换, 并交换一些列的顺序, 以式 (8.3) 为例, 将左边 $(m-k)$ 列 f 的系数依照上式减去 g 系数列的倍数, 可以消为 ρr 的系数列, 将 ρr 的 $(m-k)$ 个系数列向右平移 $(n-k)$ 列, 移动到右边, 与 g 的系数列交换. 于是我们可得到

$$\sigma_k = \det S_k = (-1)^{(n-k)(m-k)} \det \begin{pmatrix} 1 & & & & & \\ g_{m-1} & \ddots & & \rho r_d & & \\ \vdots & & 1 & \vdots & \ddots & \\ \vdots & & \vdots & \vdots & & \rho r_d \\ g_{2k-n+1} & \cdots & g_k & \rho r_{2k-m+1} & \cdots & \rho r_k \end{pmatrix}.$$

上面的矩阵用分块可表示为如下形式:

$$\begin{pmatrix} \Lambda & 0 \\ * & S_k(g, \rho r) \end{pmatrix},$$

其中 Λ 为下三角阵, 且对角元均为 1, 从而 $\det \Lambda = 1$. 因此, $\sigma_k = (-1)^{(n-k)(m-k)} \sigma_k(g, \rho r) = (-1)^{(n-k)(m-k)} \rho^{m-k} \sigma_k(g, r)$. □

进而我们有下面的定理:

定理8.7. 设 $f = \rho_0 r_0, g = \rho_1 r_1 \in F[x]$, $\deg f = n \geq \deg g = m$, 且 r_0, r_1 首一, 则:

1. 对于 $k(0 \leq k \leq m)$, 有

$$\sigma_k = \det S_k = \begin{cases} (-1)^{\tau_i} \rho_0^{m-n_i} \prod_{1 \leq j \leq i} \rho_j^{n_{j-1}-n_j}, & k = n_i, \\ 0, & k \notin \{n_i\}, \end{cases}$$

其中 $\tau_i = \sum_{1 \leq j < i} (n_{j-1} - n_i)(n_j - n_i)$.

2. $\sigma_m = \rho_1^{n-m}$, $\sigma_{n_{i+1}} = (-1)^{(n_i - n_{i+1})(n - n_{i+1} + i + 1)} (\rho_0 \cdots \rho_{i+1})^{n_i - n_{i+1}} \sigma_{n_i}$.

证明. 对于 1 中第一种情况, 只需归纳证明并且注意 $\sigma_k(r_{i-1}, r_i) = 1$. 根据推论 8.6 可得第二种情况.

对于 2, 只需化简 $\tau_{i+1} - \tau_i \pmod 2$, 我们有

$$\begin{aligned} \tau_{i+1} - \tau_i &\equiv \sum_{1 \leq j < i+1} (n_{j-1} - n_{i+1})(n_j - n_{i+1}) - \sum_{1 \leq j < i} (n_{j-1} - n_i)(n_j - n_i) \\ &\equiv n_{i-1} n_i - n_0 n_{i+1} - n_i n_{i+1} + i n_{i+1}^2 + n_0 n_i + n_{i-1} n_i - (i-1) n_i^2 \\ &\equiv n(n_i - n_{i+1}) - n_i n_{i+1} + i n_{i+1}^2 - (i-1) n_i^2 \\ &\equiv (n - n_{i+1})(n_i - n_{i+1}) + (i+1)(n_{i+1}^2 - n_i^2) \\ &\equiv (n - n_{i+1} + i + 1)(n_i - n_{i+1}) \pmod 2, \end{aligned}$$

于是可以得到(2)中的递推式. □

取 $k=0$, 则 σ_k 即为结式, 因此我们很自然地得到如下计算结式的方法:

推论8.7. *若* $\deg\gcd(f,g)\geq 1$, *则* $\operatorname{res}(f,g)=0$, *否则*

$$\operatorname{res}(f,g)=(-1)^{\tau}\rho_0^{n_1}\prod_{1\leq j\leq l}\rho_j^{n_{j-1}},$$

其中 $\tau=\sum_{1\leq j<l}n_{j-1}n_j$.

事实上, 对于 F 是 UFD 的情况, 若在该 UFD 的分式域为系数的多项式环中计算, 上面推论一般也是成立的. 我们举一个 $F=\mathbb{R}[y]$ 的例子作为说明, 计算时, 我们在 $\mathbb{R}(y)[x]$ 中计算.

例8.1. 考虑 $f=x^2-y$, $g=yx+1$, 二者的结式为:

$$\operatorname{res}_x(f,g)=1-y^3.$$

当用上面的推论来计算时, 我们有

$$\begin{array}{lll} n_0=2 & r_0=x^2-y & \rho_0=1,\\ n_1=1 & r_1=x+\dfrac{1}{y} & \rho_1=y,\\ n_2=0 & r_2=1 & \rho_2=\dfrac{1}{y^2}-y. \end{array}$$

于是结式为 $\operatorname{res}(f,g)=(-1)^{2\cdot 1}1^1\times y^2\times(1/y^2-y)^1=1-y^3$.

8.4 $\mathbb{Z}[x]$ 中的模 GCD 算法

模算法基于的思想是将整数环中的运算化为有限域上的运算, 通过中国剩余定理算法将有限域中结果还原到整数环中. 设 $f,g\in\mathbb{Z}[x]$, p 为素数, 令 $h=\gcd(f,g), h_p=h \bmod p, v_p=\gcd(f_p,g_p)$, 若有 $h=h_p=v_p$, 那么就可以用有限域中的计算结果 v_p 来代替 h. 我们需要 Mignotte 界条件来保证 $h=h_p$, 而对于 $h_p=v_p$ 成立的条件, 我们后面也会在每个算法后面具体给出理论上的分析.

8.4.1 Mignotte 界

首先我们简要说明一下为什么我们需要对多项式系数的上界作出估计. 根据前面的分析, 利用模算法求最大公因子需要保证 $h=h_p(=h \bmod p)$. 很自然地, 在取对称表示时, 只需要 h 的系数绝对值的上界比 $p/2$ 小即可. Mignotte 界即是关于整系数多项式任一非平凡因子的系数绝对值上界的一个估计. 读者可以选择跳过本小节只接阅读模算法, 只需要记住定理 8.9 的结论即可.

定义8.13 (多项式的范数)**.** 设 $f=\sum_{i=0}^{n} f_i x^i \in \mathbb{C}[x]$, 定义其 k-范数为:

$$\|f\|_k = \left(\sum_{i=0}^{n} |f_i|^k\right)^{\frac{1}{k}}.$$

注116. 由线性赋范空间的相关内容, 我们熟知有如下结论:

$$\|f\|_\infty \le \|f\|_2 \le \|f\|_1 \le (n+1)\|f\|_\infty.$$

首先我们有如下的引理:

引理8.4. 设 $f \in \mathbb{C}[x]$, $z \in \mathbb{C}$, 我们有 $\|(x-z)f\|_2 = \|(\overline{z}x-1)f\|_2$.

证明. 设 $\deg f = n$, 并记 $f_{-1} = f_{n+1} = 0$, 则

$$\begin{aligned}
\|(x-z)f\|_2^2 &= \sum_{i=0}^{n+1} |f_{i-1} - zf_i|^2 = \sum_{i=0}^{n+1} (f_{i-1} - zf_i)(\overline{f_{i-1}} - \overline{z}\overline{f_i}) \\
&= \|f\|_2^2(1+|z|^2) - \sum_{i=0}^{n+1} (z\overline{f_{i-1}}f_i + \overline{z}f_{i-1}\overline{f_i}) \\
&= \sum_{i=0}^{n+1} (\overline{z}f_{i-1} - f_i)(z\overline{f_{i-1}} - \overline{f_i}) \\
&= \sum_{i=0}^{n+1} |\overline{z}f_{i-1} - f_i|^2 \\
&= \|(\overline{z}x-1)f\|_2^2,
\end{aligned}$$

证毕. □

注117. 对于上面的引理, 我们可以给出如下更加巧妙的证明:

证明. 我们给出断言, $\forall f \in \mathbb{C}[x]$,

$$\|f\|_2^2 = \frac{1}{2\pi}\int_0^{2\pi} |f(e^{i\phi})|^2 \mathrm{d}\phi.$$

此由三角函数系的正交归一性显然可以看出. 于是

$$\begin{aligned}
\|(x-z)f\|_2^2 &= \frac{1}{2\pi}\int_0^{2\pi} |(e^{i\phi}-z)f(e^{i\phi})|^2 \mathrm{d}\phi = \frac{1}{2\pi}\int_0^{2\pi} |(ze^{-i\phi}-1)f(e^{i\phi})|^2 \mathrm{d}\phi \\
&= \frac{1}{2\pi}\int_0^{2\pi} |(\overline{z}e^{i\phi}-1)f(e^{i\phi})|^2 \mathrm{d}\phi \\
&= \|(\overline{z}x-1)f\|_2^2,
\end{aligned}$$

证毕. □

定义8.14. 设 $f \in \mathbb{C}[x]$, $z_1, z_2, \ldots, z_n$ 是 $f(x) = 0$ 的 n 个复根, 定义 $M(f) = |f_n| \prod_{i=1}^{n} \max\{1, |z_i|\}$.

由定义可知 $M(f) \geq |f_n|$ 且 $f = gh \Rightarrow M(f) = M(g)M(h)$. 关于 $M(f)$, 还有下面的 Landau 不等式成立.

定理8.8 (Landau 不等式)**.** $\forall f \in \mathbb{C}[x]$, $M(f) \leq \|f\|_2$.

证明. 不妨设 $|z_1|, |z_2|, \ldots, |z_k| > 1$ 且 $|z_{k+1}|, \ldots, |z_n| \leq 1$, 则

$$M(f) = |f_n z_1 z_2 \cdots z_k|.$$

令 $g = f_n \prod_{i=1}^{k} (\overline{z_i}x - 1) \prod_{k+1}^{n} (x - z_i) = g_n x^n + \cdots + g_0 \in \mathbb{C}[x]$, 则由引理 8.4 有

$$\begin{aligned} M^2(f) &= |f_n \overline{z_1 z_2} \cdots \overline{z_k}|^2 = |g_n|^2 \leq \|g\|_2^2 \\ &= \left\| \frac{g}{(\overline{z_1}x - 1)} (x - z_1) \right\|_2^2 \\ &= \cdots = \left\| \frac{g}{(\overline{z_1}x - 1)(\overline{z_2}x - 1) \cdots (\overline{z_k}x - 1)} (x - z_1)(x - z_2) \cdots (x - z_k) \right\|_2^2 \\ &= \|f\|_2^2, \end{aligned}$$

证毕. □

由 Landau 不等式易得出下面的命题.

引理8.5. 若 $h = \sum_{i=0}^{m} h_i x^i \in \mathbb{C}[x]$ 整除 $f = \sum_{i=0}^{n} f_i x^i \in \mathbb{C}[x]$, $n \geq m$, 则 $\|h\|_2 \leq \|h\|_1 \leq 2^m M(h) \leq \left| \frac{h_m}{f_n} \right| 2^m \|f\|_2$.

证明. 设 $h = h_m \prod_{i=1}^{m} (x - u_i)$, 则由韦达定理,

$$h_i = (-1)^{m-i} h_m \sum_{\substack{S \subset \{1,2,\ldots,m\} \\ \#S = m-i}} \prod_{j \in S} u_j,$$

于是 $|h_i| \leq |h_m| \sum_S \prod_{j \in S} |u_j| \leq |h_m| \binom{m}{i} \prod_{i=1}^{m} |u_j| \leq \binom{m}{i} M(h)$, 故有

$$\|h\|_2 \leq \|h\|_1 = \sum_{i=0}^{m} |h_i| \leq 2^m M(h) \leq \left| \frac{h_m}{f_n} \right| 2^m M(f) \leq \left| \frac{h_m}{f_n} \right| 2^m \|f\|_2.$$

证毕. □

由此, 我们可以得到 Mignotte 界.

定理8.9 (Mignotte 界). 设 $f, g, h \in \mathbb{Z}[x]$ 且 $\deg f = n \geq 1$, $\deg g = m$, $\deg h = k$, $gh \mid f$, 则:

1. $\|g\|_\infty \|h\|_\infty \leq \|g\|_2 \|h\|_2 \leq \|g\|_1 \|h\|_1 \leq 2^{m+k} \|f\|_2 \leq (n+1)^{1/2} 2^{m+k} \|f\|_\infty$,
2. $\|h\|_\infty \leq \|h\|_2 \leq 2^k \|f\|_2 \leq 2^n \|f\|_1$,
3. $\|h\|_\infty \leq \|h\|_2 \leq (n+1)^{1/2} 2^k \|f\|_\infty$.

证明. 由引理 8.5 , 我们有

$$\|g\|_1 \|h\|_1 \leq 2^{m+k} M(g) M(h) \leq 2^{m+k} M(f) \leq 2^{m+k} \|f\|_2,$$

此即第一式. 再令 $g = 1$ 可得后两式. □

8.4.2 大素数模公因子算法

如不作特殊说明, 本小节中 $\mathbb{F}_p$ 域取对称代表元 $\mathbb{F}_p = \left\{ i \in \mathbb{Z} \mid -\frac{p}{2} < i < \frac{p}{2} \right\}$. 大素数模算法要用到 $\mathbb{F}_p$ 中 Euclid 算法, 但是素数 p 一般会很大, 尤其对于次数较高的多项式, 其 Mignotte 界会变得很大, 这样在大素数有限域中的计算未必那么划算. 事实上, 我们采用的一般都是小素数模方法, 该方法将在下一小节介绍. 尽管如此, 我们还是要介绍一下大素数模方法, 以使读者能对模方法的基本思路有一个直观的认识, 便于后面理解小素数模方法.

我们先给出大素数模算法, 再对其进行分析.

算法8.4 (大素数模公因子算法).

输入: $\mathbb{Z}[x]$ 中本原式 f, g, 且 $\deg f = n \geq \deg g \geq 1$, $\|f\|_\infty \leq A$, $\|g\|_\infty \leq A$,

输出: $h = \gcd(f, g) \in \mathbb{Z}[x]$.

1. 赋值 $b = \gcd(\mathrm{lc}(f), \mathrm{lc}(g))$, $B = (n+1)^{1/2} 2^n A b$,
2. 任取素数 $p \in (2B, 4B]$,
3. $f_p = f \bmod p$, $g_p = g \bmod p$,
4. $v_p = \gcd(f_p, g_p) \in \mathbb{F}_p[x]$,
5. 计算 $w, f^*, g^* \in \mathbb{F}_p[x]$ 使得

$$w \equiv b v_p \pmod p, \quad f^* w \equiv b f \pmod p, \quad g^* w \equiv b g \pmod p,$$

6. 若 $\|f^*\|_1\|w\|_1 \le B$ 且 $\|g^*\|_1\|w\|_1 \le B$ 则继续下步, 否则跳回第 2 步,

7. 输出 $\mathrm{pp}(w)$.

注118. 之所以在 $(2B, 4B]$ 任取素数, 可以参考 [174]18.4 节有关内容. 该处提供了在此区间中随机生成素数的方法并且作了相关复杂度的分析.

下面的定理保证了算法 8.4 中判定条件的正确性.

定理8.10. *算法 8.4 中给定的条件*

$$\|f^*\|_1\|w\|_1 \le B, \quad \|g^*\|_1\|w\|_1 \le B$$

等价于 $p \nmid r$, 且此时算法 8.4 输出正确的最大公因子.

证明. 若条件满足, 则

$$\|f^*w\|_\infty \le \|f^*w\|_1 \le \|f^*\|_1\|w\|_1 \le B < p/2.$$

同理 $\|bf\|_\infty < p/2$, 由于 $f^*w \equiv bf \pmod p$, 可知 $f^*w = bf$, 同样我们也可证明 $g^*w = bg$. 于是 $\deg w = \deg \gcd(bf, bg) \Rightarrow \mathrm{pp}(w) = \gcd(f, g)$.

另一方面, 若 $\mathrm{pp}(w) = \gcd(f, g)$, 则由定理 8.9 可推出所需条件. □

定理中所说的条件即是用来保证本节开始提到的 $h_p = v_p$ 的条件, 这从定理 8.10 的证明可以看出. 我们还可以用试除法来代替这一条件, 算法及其说明见下:

算法8.5 (大素数模公因子算法)**.**

可将算法 8.4 第 6 步中条件改为 $\mathrm{pp}(w_p) \mid f$ 且 $\mathrm{pp}(w_p) \mid g$, 第 5 步只计算 w.

算法有效性. 在 $\mathbb{Q}[x]$ 中首一多项式的意义下, 由试除条件有 $v_p \mid \gcd(f, g)$. 其次, 由 $\gcd(f, g) \mid f$ 知 $\gcd(f, g) = \gcd(f, g)_p \mid f_p$, 同样地, $\gcd(f, g) \mid g_p$. 于是, $\gcd(f, g) \mid \gcd(f_p, g_p) = v_p$, 因而 $v_p = \gcd(f, g)$. □

算法中是随机从区间 $(2B, 4B]$ 中选取一个素数, 这种随机性是否保证我们能较快地得到正确的结果呢? 下面的定理作出了回答.

定理8.11. *使得算法 8.5 条件满足的素数 p 是素数集中仅除了有限个素数后的集合.*

证明. 设 $h = \text{monic}(\gcd(f,g)), h_p = h \bmod p$. 只需证若 p 不同时整除 $\text{lc}(f)$, $\text{lc}(g)$, 且 $p \nmid \text{res}(f/h, g/h)$, 则 $h = h_p = v_p$. 首先显然有 $h_p \mid v_p$. 若 $\deg v_p = \deg h_p = 0$, 则由首一性知 $v_p = h_p$, 否则 $\deg v_p \geq 1$, 此时由 $p \nmid \gcd(\text{lc}(f), \text{lc}(g))$ 可得 $h_p \neq 0$. 而存在 s, t 使

$$\begin{aligned} & sf/h + tg/h = \text{res}(f/h, g/h) \\ \Rightarrow \quad & sf + tg = \text{res}(f/h, g/h)h \\ \Rightarrow \quad & s_p f_p + t_p g_p = \text{res}(f/h, g/h)_p h_p \\ \Rightarrow \quad & v_p \mid h_p, \end{aligned}$$

故有 $h_p = v_p$. □

8.4.3 小素数模公因子算法

小素数阶有限域上的多项式计算实现起来显然要比大素数高效很多. 本节将要提到的算法基于的思想很简单. 考虑 $h = \gcd(f,g)$, 我们随机选取一系列的小素数 $p_1, p_2, \ldots, p_k$, 在诸有限域 $\mathbb{Z}_{p_i} (1 \leq i \leq k)$ 上计算 $v_{p_i} = \gcd(f_{p_i}, g_{p_i})$, 由定理 8.11 , 我们可以期望 $h_{p_i} = v_{p_i}$ 有很大概率是成立的. 命 $m = \prod_{1 \leq i \leq k} p_i$, 利用中国剩余定理, 可以由诸多 v_{p_i} 求出 $h_m = h \bmod m$, 它满足 $h_m \equiv v_{p_i} \pmod{p_i}$. 当 m 大于 Mignotte 界时, 即有 $h = h_m$.

下面仅给出小素数模算法的两种实现. 其中算法 8.6 参考 [174], 算法 8.7 参考 [13], 两者的思想基本上是一样的. 两个算法中任取素数时都加上了限制 $p < 2k \ln k$, 此参看 [174] 生成素数的有关章节. 从算法的实现细节上来说, 第二个算法比第一个算法略优, 其能减少很多不必要的计算.

算法8.6 (小素数模公因子算法).

输入: $\mathbb{Z}[x]$ 上本原多项式 f, g, 且 $\deg f = n \geq \deg g \geq 1$, $\|f\|_\infty \leq A$, $\|g\|_\infty \leq A$,

输出: $h = \gcd(f,g) \in \mathbb{Z}[x]$.

1. 赋值 $b = \gcd(\text{lc}(f), \text{lc}(g))$, $k = \lceil 2\log_2((n+1)^n b A^{2n}) \rceil$, $B = (n+1)^{1/2} 2^n A b$, $l = \lceil \log_2(2B+1) \rceil$,
2. 任取含 $2l$ 个不同的素数的集合 S, 且 $\forall p \in S, p < 2k \ln k$,
3. $S = \{p \in S \mid p \nmid b\}$,

4. $\forall p \in S$, $v_p = \gcd(f_p, g_p) \in \mathbb{F}_p[x]$,

5. $e = \min\{\deg v_p \mid p \in S\}$, $S = \{p \in S \mid \deg v_p = e\}$,

6. 若 $|S| > l$ 则去掉 S 中 $(|S| - l)$ 个元素, 只保留 l 个, 否则转到第 2 步,

7. 用中国剩余定理计算 w, f^*, $g^* \in \mathbb{Z}[x]$ 使得(对三者取对称表示)

$$\|w\|_\infty, \|f^*\|_\infty, \|g^*\|_\infty \leq \frac{1}{2}\prod_{p\in S} p =: \frac{m}{2},$$

且 $\forall p \in S$ 有

$$w \equiv bv_p \pmod p, \quad f^*w \equiv bf \pmod m, \quad g^*w \equiv bg \pmod m,$$

8. 若 $\|f^*\|_1\|w\|_1 \leq B$ 且 $\|g^*\|_1\|w\|_1 \leq B$ 则继续下一步, 否则转到第 2 步,

9. 输出 $\mathrm{pp}(w)$.

注119. l 的选取使得 $2^l > 2B$, 因而可见 S 中 l 个元素相乘之积必大于 $2B$, 于是 $m/2 > B$.

算法有效性. 由 $\|f^*\|_1\|w\|_1 \leq B$ 可知 $\|f^*w\|_\infty \leq B < m/2$. 另一方面, 由算法第 1 步计算 B 的方法可知 $\|bf\|_\infty \leq B < m/2$. 于是

$$f^*w \equiv bf \pmod m \Rightarrow f^*w = bf,$$

同理我们也有 $g^*w = bg$, 由此 $w \mid \gcd(bf, bg) = b\gcd(f, g)$. 如果取多项式的本原部分, 很容易得到 $\mathrm{pp}(w) \mid \gcd(f, g)$. 我们已假定对于所取的 p 有 $v_p = h_p = h \bmod p$, 其中 $h = \gcd(f, g)$, 则 $\deg w = \deg v_p = \deg h$, 因而 $\mathrm{pp}(w) = h$.

反过来我们假设 $\mathrm{pp}(w) = \gcd(f, g)$ 时, 易知算法也会因为第 8 步中的条件满足而输出. □

算法8.7 (小素数模公因子算法).

输入: $\mathbb{Z}[x]$ 中本原多项式 f, g, 且 $\deg f = n \geq \deg g \geq 1$, $\|f\|_\infty \leq A$, $\|f\|_\infty \leq A$,

输出: $h = \gcd(f, g) \in \mathbb{Z}[x]$.

1. $b = \gcd(\mathrm{lc}(f), \mathrm{lc}(g))$, $B = (n+1)^{1/2}2^n Ab$, $k = \lceil 2\log_2((n+1)^n bA^{2n})\rceil$,

2. 任取素数 $p > b$(实际上 $p \nmid b$ 即可), 且 $p < 2k\ln k$,
3. $v_p = \gcd(f_p, g_p) \in \mathbb{F}_p[x]$,
4. 若 $\deg v_p = 0$ 则输出 1,
5. $p_1 = p$, $v_1 = v_p$,
6. 若 $p_1 \leq 2B$, 则执行下一步, 否则跳转第 12 步,
7. 再取素数 $p > b$(或者 $p \nmid b$), 且 $p < 2k\ln k$, $p \nmid p_1$, 令 $v_p = \gcd(f_p, g_p) \in \mathbb{F}_p[x]$,
8. 若 $\deg v_p = 0$ 则输出 1,
9. 若 $\deg v_p < \deg v_1$, 则 $p_1 = p$, $v_1 = v_p$, 并跳至第 6 步,
10. 若 $\deg v_p = \deg v_1$ 则由中国剩余定理计算 V 使得

$$V \equiv v_1 \pmod{p_1}, \quad V \equiv v_p \pmod{p},$$

并赋值 $p_1 = p_1 p$, $v_1 = V$,

11. 转回第 6 步,
12. 计算 $w = bv_1 \bmod p_1$, 若 $w \mid bf$ 且 $w \mid bg$ 则继续, 否则转回第 2 步,
13. 输出 $\mathrm{pp}(w)$.

算法的有效性同样可由 $w \mid b\gcd(f,g)$ 且 $\deg w = \deg\gcd(f,g)$ 看出, 可见前面小素数模方法和上一节大素数模方法的证明, 这里不再详细叙述了.

注120. 对于大素数和小素数模算法的复杂度, 可见 [174] 第 6 章的相关分析, 在该文献 6.13 节给出了具体实现之后算法复杂度的比较, 其中大素数模方法的复杂度为 n^4, 而小素数模方法的复杂度为 n^3([174] 表 6.4).

注121. 关于大, 小素数模方法的扩展算法(即同时计算 Bezout 系数), 这里就不再加以介绍了, 因为在后文的因子分解算法中一般只需要求最大公因子即可. 有兴趣的读者可以参考 [174]6.11 等章节.

8.5 多项式组的概率算法

下面引进又一个概率性的一元多项式 GCD 算法. 对于多项式组来说, 我们当

然可以一步一步求每两个多项式的最大公因子得到整个多项式组的最大公因子, 然而下面介绍概率算法虽然得到的是可能解, 效率却更高. 首先, 我们给出:

引理8.6. *设 R 为 UFD, $n \in \mathbb{N}$, $S \subset R$ 为有限集且令 $s = \#S$, $r \in R[x_1, \dots, x_n]$ 次数不超过 d, 那么*

1. *若 r 非零, 则 r 在 S^n 中最多有 ds^{n-1} 个零点.*
2. *若 $s > d$ 且 r 在 S^n 上取值为零, 则 $r = 0$.*

证明. 先证第一条, $n = 1$ 情形显然. 将 r 写成 x_n 的多项式 $r = \sum_{0 \le i \le k} r_i x_n^i$, 其中 $r_i \in R[x_1, \dots, x_{n-1}]$ 且 $r_k \neq 0$, 由于 $\deg r_k \le d - k$, 则由归纳假设, r_k 在 S^{n-1} 中至多有 $(d-k)s^{n-2}$ 个零点, 于是 r 和 r_k 至多在 S^n 中有 $(d-k)s^{n-1}$ 个零点. 并且 $\forall a \in S^{n-1}(r_k(a) \neq 0)$, $r_a = \sum_{0 \le i \le k} r_i(a) x_n^i \in R[x_n]$ 至多有 k 个零点, 因此零点数目上限可估为:

$$(d-k)s^{n-1} + ks^{n-1} = ds^{n-1}.$$

而由第一条可以直接得到第二条的结论. □

根据引理 8.6 可知, 若从 S^n 中随机选取 a, 则有概率 $\mathrm{P}\{r(a) = 0 \mid a \in S^n\} \le d/\#S$. 假设我们有一个有限集 $S \subset F$ 和一个 S 上随机数发生器, 则有下面的:

算法8.8 (多项式组最大公因子的概率算法).
输入: $f_1, \dots, f_n \in F[x]$, F 是域,
输出: 首一多项式 h, 且 $h = \gcd(f_1, \dots, f_n)$ 可能正确.

1. 任取 $a_3, \dots, a_n \in S$,
2. 令 $g = f_2 + \sum_{3 \le i \le n} a_i f_i$,
3. 输出 $\gcd(f_1, g)$.

注122. 实践中使用快速 Euclid 算法来提高效率.

定理8.12. *设 h 为算法 8.8 中的输出结果, $h^* = \gcd(f_1, \dots, f_n)$. 则 $h^* \mid h$, 且 $\mathrm{P}\{h \neq h^*\} \le d/\#S$.*

证明. $h^* \mid h$ 显然. 通过将 f_i 除以 h^* 我们可以假设整个函数组最大公因子为 1, 设 $f_1 \neq 0$, 令 $A_3, \dots, A_n$ 为 $F[x]$ 上新的未定元, 令 $R = F[A_3, \dots, A_n]$,

再令 $K = F(A_3, \ldots, A_n)$ 为 R 的分式域, 命 $G = f_2 + \sum_{3\leq i\leq n} A_i f_i \in R[x]$, $r = \mathrm{res}_x(f_1, G) \in R$, 则 r 是关于 $A_3, \ldots, A_n$ 的次数不超过 d 的多项式. 令理想 $I = \langle A_3 - a_3, \ldots, A_n - a_n\rangle$, 有 $R/I \cong F$, 由引理 8.2 有

$$\overline{r} = r(a_3, \ldots, a_n) \neq 0 \Leftrightarrow \mathrm{res}_x(f_1, g) \neq 0 \Leftrightarrow \gcd(f_1, g) = 1,$$

设 u 为 f_1, G 在 $R[x]$ 中的公因子, 由于 $u \mid f_1$, 知它的系数都属于 f_1 在 F 的分裂域 E, 但是 $E[x] \cap K[x] = F[x]$, 因此 $u \in F[x]$. 于是由 $u \mid G$ 推知 $u \mid f_i$ $(i = 1, \ldots, n)$, 又由于 $\gcd(f_1, \ldots, f_n) = 1$, 知 $u \in F$, 因此 $\gcd(f_1, G)$ 为一常数, r 是 R 中一非零元. 由引理 8.6 可得定理中对于概率的估计. □

实际过程中, 可以取 f_1 为最小次数的多项式, 以使错误概率降低, 并且在算法结束后检验 h 是否为所求的最大公因子, 若不是, 则重新选取随机数计算, 直至输出正确结果.

9 有限域上多项式因子分解

多项式因子分解问题比最大公因子问题要复杂也更困难一些. 然而令人惊喜的是, 相对于整数上多项式来说, 在有限域上多项式的因子分解问题却较为简单. 这给我们提供了一种将整数环或有理数域上的多项式因子分解问题转化到较简单的有限域情况上来解决的可能性.

这一部分我们首先解决 $\mathbb{F}_p$ 域上的多项式因子分解问题. 这些内容是 $\mathbb{Z}[x]$, $\mathbb{Q}[x]$, $\mathbb{Z}_m[x]$ 乃至 $\mathbb{R}[x]$, $\mathbb{C}[x]$ 和多元多项式因子分解的基础.

在有限域上进行因子分解的方法很多, 一般来说, 有限域上多项式因子分解要经过下面三个步骤:

1. 无平方因子分解(Squarefree Factorization)
2. 不同次因子分解(Distinct-degree Factorization)
3. 同次因子分解(Equal-degree Factorization)

其中第 2, 3 步或者单独第 3 步可以由 Berlekamp 算法代替. 这一章首先将从上面三个方面介绍有限域上的因子分解问题, 然后讨论 Berlekamp 算法.

本章我们假定读者具有有限域及群论等基本知识, 这方面可以参考相关的抽象代数或数论等书籍. 当然, 我们下面也会对一些较重要的知识作一些简单介绍.

9.1 不同次数因子分解

首先我们假定多项式是无平方因子的(Squarefree), 即无重因子. 这一点很容

易做到, 如果 f 含重因子, 那么取 $f/\gcd(f,f')$, 则可消去重因子, 得到无重因子的多项式. 不同次因子分解即是在无平方因子分解的基础上, 将多项式中各不同次数因子的乘积逐一剥离出来. 这一算法, 最早出现在 Zassenhaus[189] 的论述中.

首先我们介绍一些有限域的准备知识, 它们对于理解本章的算法是必要的.

9.1.1 有限域 $\mathbb{F}_p$ 和 $\mathbb{F}_{p^d}$

在数论教程中, 我们熟知如下的 Fermat 小定理.

定理9.1 (Fermat 小定理). *对于非零元 $a\in\mathbb{F}_p$, 我们有 $a^{p-1}=1$. 对于 $a\in\mathbb{F}_p$, 我们有 $a^p=a$, 且*

$$x^p-x=\prod_{a\in\mathbb{F}_p}(x-a).$$

注123. 我们知道, 有限域的阶数只可能是素数以及素数的幂, 对于 $d\geq 1\in\mathbb{N}$, 可以构造 p^d 阶的域如下:

$$\mathbb{F}_{p^d}=\mathbb{F}_p[x]/\langle f\rangle,$$

其中 $f\in\mathbb{F}_p[x]$ 是 d 次不可约多项式.

注124. Fermat 小定理 p 对于素数以及素数的幂均成立, 素数的幂情况证明同素数情况.

下面的定理是 Fermat 小定理的推广, Fermat 小定理是其 $d=1$ 的特殊情形.

定理9.2 (Fermat 小定理推广). *对于任何 $d\geq 1$, $x^{p^d}-x\in\mathbb{F}_p[x]$ 是 $\mathbb{F}_p[x]$ 中所有次数整除 d 的不可约首一多项式的乘积. 其中 p 是素数或素数幂.*

证明. 由 Fermat 小定理知 $h=x^{p^d}-x$ 是所有的一次因子 $x-a$ 的乘积, 其中 a 取遍 $\mathbb{F}_{p^d}$ 中的元素. 因此, h 是无平方因子的, 于是我们只须证明对任何 $\mathbb{F}_p[x]$ 中首一不可约 n 次多项式 f:

$$f\mid x^{p^d}-x\Leftrightarrow n\mid d.$$

充分性: 若 $f\mid x^{p^d}-x$, 则由 Fermat 小定理, 可以取 $\mathbb{F}_{p^d}$ 的子集 A 使得 $f=\prod_{a\in A}(x-a)$. 任取 $a\in A$, 令 $\mathbb{F}_p[x]/\langle f\rangle\cong\mathbb{F}_p(a)\subset\mathbb{F}_{p^d}$, 其中 $\mathbb{F}_p(a)$ 是 $\mathbb{F}_{p^d}$ 中包含 a 的最小子域, 有 p^n 个元素, $\mathbb{F}_{p^d}$ 是它的扩域, 因此存在正整数 e 使得 $p^d=(p^n)^e$, 即有 $n\mid d$.

必要性: 若 $n\mid d$, 令 $\mathbb{F}_{p^n}=\mathbb{F}_p[x]/\langle f\rangle$, 且 $a=(x \bmod f)\in\mathbb{F}_{p^n}$ 为 f 的一个根. 于是 $a^{p^n}=a\Rightarrow(x-a)\mid(x^{p^n}-x)$, 由于 $p^n-1\mid p^d-1$, 设

$$p^d-1=(p^n-1)e=(p^n-1)(p^{d-n}+p^{d-2n}+\cdots+1),$$

则

$$x^{p^d-1}-1=(x^{p^n-1}-1)(x^{(p^n-1)(e-1)}+\cdots+1).$$

将上式乘以 x 则可得 $(x-a)\mid(x^{p^n}-x)\mid(x^{p^d}-x)$. 因此在 $\mathbb{F}_{p^n}[x]$ 中 $(x-a)\mid \gcd(f,x^{p^d}-x)$, 由于 $\mathbb{F}_p[x]$ 中多项式的最大公因子应该也在 $\mathbb{F}_p[x]$ 中, 于是由其非平凡可推出 $\gcd(f,x^{p^d}-x)=f$, 即 $f\mid x^{p^d}-x$. □

9.1.2 不同次因子分解

不同次因子分解算法的目标即是要求出多项式的不同次因子序列, 其定义如下:

定义9.1 (不同次因子序列). $\mathbb{F}_p[x]$ 中非平凡多项式 f 的不同次因子分解是指得到如下的不同次因子序列 $(g_1,\ldots,g_s)$, 其中 g_i 是 f 在 $\mathbb{F}_p[x]$ 中所有首一不可约 i 次多项式的乘积, 且 $g_s\neq 1$.

由定理 9.2 , 我们很容易构造出如下不同次因子分解算法.

算法9.1 (不同次因子分解).

输入: 无平方因子 $n(n>0)$ 次首一多项式 $f\in\mathbb{F}_p[x]$,

输出: f 的不同次因子序列 $(g_1,\ldots,g_s)$.

1. $h_0=x$, $f_0=f$, $i=0$,
2. $i=i+1$, 在环 $R=\mathbb{F}_p[x]/\langle f\rangle$ 中调用快速求幂算法(例如算法 4.1)计算 $h_i=h_{i-1}^p \bmod f$,
3. $g_i=\gcd(h_i-x,f_{i-1})$, $f_i=\dfrac{f_{i-1}}{g_i}$,
4. 若 $f_i\neq 1$ 则转到第 2 步,
5. $s=i$, 输出 $(g_1,\ldots,g_s)$.

算法有效性. 设 $(G_1,\ldots,G_t)$ 是 f 的不同次因子序列, 考虑命题

$$P_i: h_i\equiv x^{p^i}\pmod f,\quad f_i=G_{i+1}\cdots G_t,\quad g_i=G_i(\text{若}i>0).$$

显然 P_0 成立, 设 $P_0,\ldots,P_{i-1}$ 均成立, 则对 $i\geq 1$, 有 $h_i\equiv h_{i-1}^p\equiv x^{p^i}\pmod f$ 且

$$g_i=\gcd(h_i-x,f_{i-1})=\gcd(x^{p^i}-x,f_{i-1}).$$

由定理 9.2 , g_i 是 $\mathbb{F}_p[x]$ 中所有首一不可约且次数整除 i 的多项式的乘积且能整除 $f_{i-1}=G_i\cdots G_t$, 因此 $g_i=G_i$(低于 i 次的因子已在前面提出了). 于是归纳证明了 $P_i(0\le i\le s)$ 成立. 同时也可得到 $s=t$. □

注125. 算法 9.1 可在 $\deg f_i<2(i+1)$ 时即终止, 因为 f_i 所有不可约因子次数至少为 $i+1$, 因此 f_i 已经是不可约的了.

注126. 算法 9.1 第 2 步求 $h_{i-1}^p \bmod f$ 时也可利用 $\mathbb{F}_p$ 中 p 次幂的特殊性质来计算, 即对于 $f=a_0+a_1x+\cdots+a_nx^n$, 有

$$f^p=a_0+a_1x^p+\cdots+a_nx^{np}.$$

例9.1. 考虑多项式 $f=x^5+x^3+x^2+x-1\in\mathbb{F}_3[x]$.

解: 我们将单步执行算法 9.1 的步骤列于下:

$$\begin{aligned}
f_0&=f=x^5+x^3+x^2+x-1,\\
h_0&=x,\\
h_1&=h_0^3 \bmod f=x^3 \bmod f=x^3,\\
g_1&=\gcd(h_1-x,f_0)=x^2-1,\\
f_1&=\frac{f_0}{g_1}=x^3-x+1,\\
h_2&=h_1^3 \bmod f=x^9 \bmod f=-x^3-x,\\
g_2&=\gcd(h_2-x,f_1)=1,\\
f_2&=x^3-x+1.
\end{aligned}$$

此时算法已可以终止, 得到不同次因子序列为 $(x^2-1,1,x^3-x+1)$. ◇

不同次因子分解算法利用快速求幂算法以及快速 Euclid 算法, 其复杂度可以达到 $O(sM(n)\log(nq))$ 次 $\mathbb{F}_p$ 中的运算([174] 定理 14.4), 其中 s 是 f 不可约因子最大的次数.1998 年, Kaltofen 和 Shoup[97] 提出了一种改进的不同次因子分解算法(Baby step-Giant step 算法), 其复杂度达到了 $O(n^{1.815}\log p)$, 当有限域的阶 p 小于多项式次数 n 时, 该算法的效率较高.

9.2 同次因子分解

本节对上一节不同次因子分解的结果继续进行同次因子分解, 或称为 Cantor-Zassenhaus 算法[50], 最终将 f 完全分解为不可约多项式的积. 由于域的特征为奇素数与 2 的两种情况在处理中有一点微小的技术区别, 因此本节分成两部分来讨论.

9.2.1 特征为奇素数的有限域

首先, 我们给出一些必要的群论方面的准备命题.

引理9.1. 设 q 为一素数幂, k 为 $q-1$ 的因子, 记 $\mathbb{F}_q^*$ 为 $\mathbb{F}_q$ 中除去零元素所构成的乘法群, $S=\{b^k \mid b\in\mathbb{F}_q^*\}$ 为 $\mathbb{F}_q^*$ 中的 k 次幂集合, 则:

1. S 为 $(q-1)/k$ 阶子群.
2. $S=\{a\in\mathbb{F}_q^* \mid a^{(q-1)/k}=1\}$.

证明. S 为 k 次幂同态映射 $\sigma_k:\mathbb{F}_q^*\to\mathbb{F}_q^*$ 的象, 从而为 $\mathbb{F}_q^*$ 的子群. σ_k 的核为 k 次单位根:

$$\ker\sigma_k=\{a\in\mathbb{F}_q^* \mid \sigma_k(a)=1\}.$$

由于 $\mathbb{F}_q$ 为域, 可知多项式 $x^k-1\in\mathbb{F}_q[x]$ 的根至多有 k 个, 于是 $\#\ker\sigma_k\le k$.

因为 $(b^k)^{(q-1)/k}=b^{q-1}=1$ 对于任何 $b\in\mathbb{F}_q^*$ 均成立, 由 Fermat 小定理得 $S\subset\ker\sigma_{(q-1)/k}$, 可知 $\#S\le(q-1)/k$. 于是由群同态定理得

$$q-1=\#\mathbb{F}_q^*=\#\ker\sigma_k\cdot\#\mathrm{im}\sigma_k=\#\ker\sigma_k\cdot\#S\le k(q-1)/k=q-1,$$

可知 $\#\ker\sigma_k=k$ 且 $\#S=(q-1)/k$, $S=\ker\sigma_{(q-1)/k}$. □

作为引理 9.1 的推论, 我们有

引理9.2. 设 q 为一奇素数幂, $S=\{a\in\mathbb{F}_q^* \mid \exists b\in\mathbb{F}_q^*(a=b^2)\}$, 则

1. $S\subset\mathbb{F}_q^*$ 是 $(q-1)/2$ 阶子群.
2. $S=\{a\in\mathbb{F}_q^* \mid a^{(q-1)/2}=1\}$.
3. $\forall a\in\mathbb{F}_q^*, a^{(q-1)/2}\in\{1,-1\}$.

下面的概率性算法给出 f 的可能因子, 其中 f 是经上节算法给出的无平方首一同次因子乘积, 即存在 $n=\deg f$ 的一个因子 d 使得 f 可分解为 $r=n/d$ 个 d 次首一不可约因子 $f_1,\dots f_r$.

算法9.2 (同次因子分解概率算法).
输入: f, d,
输出: f 的首一因子 g, 或者失败.

1. 随机选取 $a \in \mathbb{F}_q[x]$ 使得 $\deg a < n$. 若 $a \in \mathbb{F}_q$ 则输出失败,
2. $g_1 = \gcd(a, f)$, 若 $g_1 \neq 1$ 且 $g_1 \neq f$ 则输出 g_1,
3. 调用快速求幂算法在环 $R = \mathbb{F}_q[x]/\langle f\rangle$ 中计算 $b = a^{(q^d-1)/2} \bmod f$,
4. $g_2 = \gcd(b-1, f)$, 若 $g_2 \neq 1$ 且 $g_2 \neq f$ 则输出 g_2, 否则输出失败.

设 $f = f_1 f_2 \cdots f_r$, 则由中国剩余定理有如下的环同构:

$$\chi : R = \mathbb{F}_q[x]/\langle f\rangle \to \mathbb{F}_q[x]/\langle f_1\rangle \times \cdots \times \mathbb{F}_q[x]/\langle f_r\rangle = R_1 \times \cdots \times R_r,$$

其中 $\mathbb{F}_{q^d} \cong R_i = \mathbb{F}_q[x]/\langle f_i\rangle \supset \mathbb{F}_q$. 引入下面记号:

$$\chi(a \bmod f) = (a \bmod f_1, \ldots, a \bmod f_r) = (\chi_1(a), \ldots, \chi_r(a)),$$

其中 $\chi_i(a) = a \bmod f_i$. 记 $e = (q^d-1)/2$, 则对任意 $\beta \in R_i^* = \mathbb{F}_{q^d}^*$, 我们有 $\beta^e \in \{-1, 1\}$, 且等概率地取两个值之一. 如果我们随机任意选取 $a \in \mathbb{F}_q[x]$ 使得 $\deg a < n$ 且 $\gcd(a, f) = 1$, 则 $\chi_1(a), \ldots, \chi_r(a)$ 是 $\mathbb{F}_{q^d}^*$ 中随机元素, 且 $\varepsilon_i = \chi_i(a^e) \in R_i$ 等概率取 1 或 -1, 因此

$$\chi(a^e - 1) = (\varepsilon_1 - 1, \ldots, \varepsilon_r - 1),$$

此时 $a^e - 1$ 是 f 的一个因子(不一定不可约), 除非 $\varepsilon_1 = \cdots = \varepsilon_r$. 因为若只有部分 $\varepsilon_i = 1 (i \in T \subset \{1, 2, \ldots, r\})$, 则 $\chi_i(a^e - 1) = 0 \Rightarrow f_i \mid \gcd(a^e - 1, f)$, 那么 $\gcd(a^e - 1, f) = \prod_{i\in T} f_i$ 给出 f 的一个非平凡因子. 因此算法发生错误的概率为 ε_i 全相等的概率, 即 $2 \cdot (1/2)^r = 2^{-r+1} \leq 1/2$, 这里一般有 $r \geq 2$.

算法9.3 (同次因子分解).

输入: f, d,

输出: f 在 $\mathbb{F}_q[x]$ 中的首一不可约因子.

1. 若 $n = d$ 则输出 f,
2. 重复调用算法 9.2 , 输入 f 和 d, 直至返回 f 的一个因子 g,
3. 递归调用本算法, 分别求出 g 和 f/g 分解的结果并输出所有的结果.

例9.2. 我们讨论 $f = x^4 + x^3 + x - 1 \in \mathbb{F}_3[x]$, $d = 2$(因为我们可以假定这里的 f 是上节因子算法已分解出的 g_2, 从而此处可设 $d = 2$).

解: 随机取 $a = x$, 则 $g_1 = \gcd(a, f) = 1$,

$$b = a^{(3^2-1)/2} \bmod f = a^4 \bmod f = -x^3 - x + 1,$$

$$g_2 = \gcd(b - 1, f) = x^2 + 1,$$

我们找到了一个因子 $x^2 + 1$, 另一个因子为 $f/(x^2 + 1) = x^2 + x + 2$. ◇

注127. 算法 9.2 的复杂度为 $O((d \log q + \log n)M(n))$ 次 $\mathbb{F}_q$ 中运算([174] 定理 14.9), 而同次因子分解算法 9.3 的复杂度为 $O((d \log q + \log n)M(n) \log r)$ 次 $\mathbb{F}_q$ 中运算([174] 定理 14.11).

注128. Gathen 和 Shoup[175] 提出了一种 Frobenius 迭代算法(iterated Frobenius algorithm), 对于环 $R = \mathbb{F}_q[x]/\langle f \rangle$ 中的元素 α, 能够快速地求出序列 $\alpha, \alpha^q, \ldots, \alpha^{q^d}$, 其中 d 不超过无平方因子多项式 f 的次数 n. 该算法复杂度约为 $O(M(n)^2 \log n \log d)$ 个 $\mathbb{F}_q$ 中运算. 将其应用到不同次因子分解的求幂计算上时, 复杂度为 $O(M(n^2) \log n + M(n) \log q)$. 在同次因子分解中, 由于求幂可用下式计算:

$$\alpha^{(q^d-1)/2} = \left(\alpha^{q^{d-1}} \cdots \alpha^q \alpha\right)^{(q-1)/2},$$

因此, 其复杂度变为 $O((M(nd)r \log d + M(n) \log q) \log r)$. 以上复杂度的分析均可参考 [174]14.7 节相关定理.

9.2.2 特征为 2 的有限域

注意到引理 9.1 和引理 9.2 均是在奇素数阶有限域中成立的, 对于阶为 2 及其幂次的有限域, 上一小节提到的算法已不适用, 因此这里我们要单独讨论一下.

以下设 $\mathbb{F}_q$ 是一特征为 2 的域, 且 $q = 2^k$, k 是一正整数. $\mathbb{F}_q$ 上多项式 f 是无平方因子 n 次多项式, 且是 r 个 d 次不可约多项式之积.

定义9.2. 对于正整数 m, 定义 $\mathbb{F}_2$ 上 m 阶迹多项式(mth trace polynomial)T_m 为

$$T_m = x^{2^{m-1}} + x^{2^{m-2}} + \cdots + x^4 + x^2 + x.$$

注129. $x^{2^m} + x = T_m(T_m + 1)$. 此式可直接验证.

注130. $\forall \alpha \in \mathbb{F}_{2^m}$, $T_m(\alpha) = 0$ 或 1, 且各有一半概率. 这是因为 α 一定是 $x^{2^m} + x$ 的根, 因而 $T_m(\alpha) = 0$ 或 $T_m(\alpha) + 1 = 0$, 且二者次数均为 2^{m-1}, 因此各有 2^{m-1} 个根.

引理9.3. 设 a 是 $R=\mathbb{F}_q[x]/\langle f\rangle$ 上随机选取的一个多项式, $b=T_{kd}(a)$, 则 $\gcd(b-1,f)$ 可能给出 f 的一个非平凡因子, 且失败概率不超过 *1/2*.

证明. 首先, $\chi_i(b)=\chi_i(T_{kd}(a))=T_{kd}(\chi_i(a))=0$ 或 1, 而由中国剩余定理所得到的环同构可知, 当且仅当 $\chi_i(b)(1\le i\le r)$ 同时为 0 或 1 时, $b\in\mathbb{F}_2$, 此概率为 $2\times 2^{-r}=2^{1-r}\le\frac{1}{2}$. □

算法9.4 (特征为 2 的域上同次因子分解).

算法基本同算法 9.2 和 9.3 , 只需将算法 9.2 中第 3 步改为计算 $b=T_{kd}(a)$ 即可.

例9.3. 作为例子, 我们计算 $f=(x^3+x^2+1)(x^3+x+1)=x^6+x^5+x^4+x^3+x^2+x+1\in\mathbb{F}_2[x]$ 的同次因子分解, 其中 $d=3$, $k=1$.

解: 若取 $a=x$, 则 $b=T_2(a)=x^4+x^2+x \bmod f=x^4+x^2+x$, 而

$$g=\gcd(b,f)=x^3+x+1,$$

故 $f_1=x^3+x+1$, $f_2=f/f_1=x^3+x^2+1$. ◇

9.3 一个完整的因子分解算法及其应用

对于一般有重因子的多项式 $f\in\mathbb{F}_q[x]$, 利用上两节的方法可以完全将其分解. 设 f 是首一的, 并且 f 的首一不可约因子分解为 $f=\prod_{i=1}^m g_i^{e_i}$, 记 $U=\{(g_1,e_1),\ldots,(g_m,e_m)\}$ 表示它的分解, 则可利用下面的算法 9.5 给出此分解:

算法9.5 (一个完整的分解算法).

输入: 首一多项式 $f\in\mathbb{F}_q[x]$, q 为一素数幂.

输出: f 的分解 U.

1. $h_0=x, f_0=f, i=0, u=\varnothing$,
2. $i=i+1$,
3. (不同次因子分解)利用快速求幂算法计算 $h_i=h_{i-1}^q \bmod f$, $g=\gcd(h_i-x,f_{i-1})$,
4. 若 $g\ne 1$ 则利用算法 9.3 求出 g 的所有同次首一不可约因子 $g_1,g_2,\ldots,g_s$,

5. $f_i = f_{i-1}$, 并不断除以 g 的同次因子求出 $g_1, \ldots, g_s$ 的次数 $e_1, \ldots, e_s$, 每求出一个 e_i, 将 (g_i, e_i) 添入 u,
6. 若 $f_i \neq 1$ 则转第 2 步,
7. 输出 U.

注131. 算法 9.5 中每次循环的过程中, 都会先利用不同次因子分解求出 f 中所包含的 i 次不可约因子, 这些不可约因子的(一次)乘积即为 g, 然后依次求出 f 中包含 g 的不可约因子的次数. 此算法用试除法代替了无平方因子分解, 以求得不可约因子在待分解多项式中的重数.

作为前面提过的诸多因子分解算法的应用, 下面讨论 $\mathbb{F}_q[x]$ 中的多项式求根问题. 设 f 为 $\mathbb{F}_q[x]$ 上一非平凡多项式, 下面的算法 9.6 给出其所有 $\mathbb{F}_q$ 中的根.

算法9.6 ($\mathbb{F}_q$ 上多项式求根算法).

利用快速求幂算法(例算法如 4.1)求出 $h = x^q \bmod f$, 令 $g = \gcd(h - x, f)$, 若 $\deg g = 0$ 则无根, 否则利用同次因子分解算法 9.3 其出所有不可约因子 $x - u_1, \ldots, x - u_r$, 则 $u_1, \ldots, u_r$ 即为其所有 $\mathbb{F}_q$ 上的根.

算法避免了将 f 完全分解来求根, 求根实际上只要求出所有一次不可约因子, 因此先将其与 $x^q - x$ 取最大公因子. 由此而衍生出下面的 $\mathbb{Z}[x]$ 中求整数根的算法. 引入该算法之前, 我们先证明

引理9.4. 设 $\mathbb{Z}[x]$ 上非平凡 n 次多项式 f, $\|f\|_\infty = A$, 且 $u \in \mathbb{Z}$ 是 f 的非零根, $f = (x - u)g$, 则 $\|g\|_\infty \leq nA$.

证明. 设 $g = \sum_{i=0}^{n-1} g_i x^i$, 则 $f = (x-u)g = g_{n-1}x^n + (g_{n-2} - ug_{n-1})x^{n-1} + \cdots + (g_0 - ug_1)x - ug_0$. 式中每项系数绝对值均不超过 A, 于是

$$
\begin{aligned}
|g_0| &\leq \frac{A}{|u|}, \\
|g_1| &\leq \frac{A + |g_0|}{|u|} \leq \frac{2A}{|u|}, \\
&\vdots \\
|g_{n-1}| &\leq \frac{nA}{|u|}.
\end{aligned}
$$

由 $|u| \geq 1$ 可知 $\|g\|_\infty \leq nA$. □

算法9.7 (整数多项式整数根算法).

输入: 非平凡 n 次多项式 $f \in \mathbb{Z}[x]$, 且 $\|f\|_\infty = A$.

输出: 中 f 的不同的整数根.

1. $B = 2n(A + A^2)$, 任取奇素数 $p \in (B+1, 2B]$.
2. 使用算法 9.6 找出 $\mathbb{F}_p[x]$ 上 $f \bmod p$ 的所有根 $\{u_1 \bmod p, \ldots, u_r \bmod p\}$, 其中 $u_i \in \mathbb{Z}$ 且 $|u_i| < p/2 (1 \leq i \leq r)$.
3. 对于每个 i, 计算 $n-1$ 次多项式 $v_i \in \mathbb{Z}[x]$ 且 $\|v_i\|_\infty \leq p/2$ 使得 $f \equiv (x-u_i)v_i \pmod p$.
4. 输出 $\{u_i \mid 1 \leq i \leq r, |u_i| \leq A, \|v_i\|_\infty \leq nA\}$.

算法有效性. 不妨设 f 无零根, 则对于其任何一个非零根 $u \in \mathbb{Z}$, 其能整除 f 的常数项, 因而 $|u| \leq A < p/2$, 因此所有整数根都可以从其在模 p 下的象还原出来. 现在我们只需证明 $f(u_i) = 0$ 当且仅当 $|u_i| \leq A$ 且 $\|v_i\|_\infty \leq nA$.

若 $f(u_i) = 0$, 则显然有 $|u_i| \leq A$. 可设 $f/(x-u_i) = g$, 则由引理 9.4 知 $\|f/(x-u_i)\|_\infty = \|g\|_\infty \leq nA < p/2$. 但由于 $f/(x-u_i) \equiv v_i \pmod p$, 且两边多项式系数均比 $p/2$ 小, 知有 $f/(x-u_i) = v_i$, 故也有 $\|v_i\|_\infty \leq nA$.

另一方面, 若 $|u_i| \leq A$ 且 $\|v_i\|_\infty \leq nA$, 则 $\|(x-u_i)v_i\|_\infty \leq (1+A)nA < p/2$, 因此 $f \equiv (x-u_i)v_i \pmod p$, 可知 $f = (x-u_i)v_i$. □

9.4 无平方因子分解

这一小节详细介绍无平方因子分解, 我们分特征为零的域和有限域上多项式这两种情况进行介绍.

9.4.1 特征为零的域上无平方分解

定义9.3. 多项式 $f = \sum_{i=0}^{n} f_i x^i \in \mathbb{F}[x]$($\mathbb{F}$ 可以是环, 域等)的形式微商定义为:

$$f' = \sum_{i=0}^{n} i f_i x^{i-1}.$$

我们先假定 $\mathbb{F}$ 是一特征为零的域, 那么我们已经知道若 $u \in \mathbb{F}$ 是 f 的 m 阶零点, 则其是 f' 的 $m-1$ 阶零点, 于是 f/f' 中将只含 $(x-u)$ 的一次因子. 我们可以在任一域 $\mathbb{F}$ 内证明如下的命题:

定理9.3. $\mathbb{F}$ 是任一域, 若 $g \in \mathbb{F}[x]$ 是 $f \in \mathbb{F}[x]$ 的一不可约因子, 且 $f = g^e h$, $h \in \mathbb{F}[x]$, g, h 互素, 则有 $g^{e-1} \mid f'$, 并且 $g^e \nmid f'$ 当且仅当 $eg' \neq 0$.

证明. 由 f 的表达式我们得到

$$f' = ehg^{e-1}g' + g^e h',$$

则显然 $g^{e-1} \mid f'$. 另一方面 g^e 对 f' 的整除性等价于对 $ehg^{e-1}g'$ 的整除性, 即 g 是否能整除 eg', 由于 $\deg eg' < \deg g$, 则 $g \mid eg' \Leftrightarrow eg' = 0$. □

注132. 我们注意到, 在特征为零的域中, 当 g 是一非平凡多项式时, $g' \neq 0$, 但在特征有限的域中, 这一点并不一定正确, 如 $g = x^3 + 1 \in \mathbb{F}_3[x]$ 的形式微商 $g' = 0$. 这正是我们下面将要提到的算法不适合于有限域上的原因.

有了上面的定理, 则首先我们可以在特征为零的域上求出 f 的无平方因子部分.

算法9.8 (无平方因子部分).

对于输入的特征为零的域 $\mathbb{F}$ 上的 n 次首一多项式 f, 输出无平方因子部分 $\dfrac{f}{\gcd(f, f')}$.

因子分解时, 我们往往不仅需要求出无平方因子部分, 还要求出下面所谓的无平方因子分解. 即若首一非平凡多项式 $f = g_1 g_2^2 \cdots g_m^m$, 其中 $g_1, \ldots, g_m$ 两两互素且无平方因子, $g_m \neq 1$, 则称 $(g_1, \ldots, g_m)$ 为 f 的无平方分解(Squarefree Decompositon). Yun[188] 提出了一种较快的算法以求出无平方分解.

算法9.9 (无平方分解).

1. $u = \gcd(f, f')$, $v_1 = f/u$, $w_1 = f'/u$, $i = 1$,
2. $h_i = \gcd(v_i, w_i - v_i')$, $v_{i+1} = v_i/h_i$, $w_{i+1} = (w_i - v_i')/h_i$, $i = i + 1$,
3. 若 $v_i \neq 1$ 则转第 2 步,
4. 输出 $(h_1, \ldots, h_{i-1})$.

注133. 上述两个算法的复杂度均为 $O(M(n)\log n)$(参见 [174]14.6 节).

注134. 该算法的思想是简单的, 但要给出严格证明比较繁琐. 其基本思想是利求导运算将不同次幂的不可约因子的次数变成某项前的系数, 利用系数的不同将这些因子一层一层“剥离”出来. 这个思想对于理解后面有限域上无平方分解算法的原理是很重要的. 下面我们用一个例子来具体展示这一过程.

例9.4. 求 $f = abc^2d^3$ 的无平方分解.

解: 顺次计算可得:

$$
\begin{aligned}
f' &= a'bc^2d^3 + ab'c^2d^3 + 2abcc'd^3 + 3abc^2d^2d', & w_2 - v_2' &= cd', \\
u &= \gcd(f, f') = cd^2, & h_2 &= \gcd(cd, cd') = c, \\
v_1 &= f/u = abcd, & v_3 &= v_2/h_2 = d, \\
w_1 &= f'/u = a'bcd + ab'cd + 2abc'd + 3abcd', & w_3 &= (w_2 - v_2')/h_2 = d', \\
w_1 - v_1' &= abc'd + 2abcd', & w_3 - v_3' &= 0, \\
h_1 &= \gcd(abcd, abc'd + 2abcd') = ab, & h_3 &= \gcd(d, 0) = d, \\
v_2 &= v_1/h_1 = cd, & v_4 &= v_3/h_3 = 1, \\
w_2 &= (w_1 - v_1')/h_1 = c'd + 2cd', & w_4 &= (w_3 - v_3')/h = 0.
\end{aligned}
$$

最后输出为 (ab, c, d). ◇

9.4.2 特征有限的域上无平方分解

我们再来考虑特征有限的域上无平方分解. 在注 132 中我们已经看到特征有限与特征为零的域的区别为对于一个非平凡的多项式 $f(\deg f \geq 1)$, 它的形式微商仍然可能是零. 下面探讨一下什么情况下形式微商为零, 以及此时的多项式有什么特点.

考虑多项式 $f = \sum_{i=0}^{n} f_i x^i \in \mathbb{F}_p[x]$, p 是一素数. 若其微商为零, 则 $f' = \sum_{i=0}^{n} i f_i x^{i-1} = 0$, 若 $f_i \neq 0$, 则须有 $i \bmod p = 0$, 于是 f 中所含的单项均是 x 的 p 的倍数的幂次项, 亦即 $f = \sum_{i=0}^{n/p} f_i x^{ip}$. 于是

$$
f = \sum_{i=0}^{n/p} (f_i x^i)^p = \left(\sum_{i=0}^{n/p} f_i x^i \right)^p .
$$

即 f 是一个多项式的 p 次幂, 注意到上一小节提到的例子 $g = x^3 + 1 = (x+1)^3$. 对于素数幂阶的域 $\mathbb{F}_q (q = p^d)$, 也有下面的结论:

定理9.4. *设 $q = p^d$ 为素数 p 的幂, 非平凡多项式 $f \in \mathbb{F}_q[x]$ 且 $f' = 0$, 那么 f 为一 p 次幂.*

注135. 该定理的证明要点在于注意到同 $\mathbb{F}_p$ 一样, $\mathbb{F}_q$ 中的元 a 均有 p 次根.

推论9.1. $\mathbb{F}$ 为任一域, 则其上非平凡多项式 f 是无平方因子的当且仅当 $\gcd(f, f') = 1$.

由上面的定理我们知道不可约非平凡多项式的形式微商一定非零. 现在前面的算法唯一不可行之处即是对于 $p \mid e_i$ 的情形. 设 $f = \prod_{i=1}^{r} f_i^{e_i}$, 式中 f_i 两两互素且不可约, $\deg f = n \geq 1$, $e_1, \ldots, e_r$ 均是正整数. 显然我们有

$$f' = \prod_{p \nmid e_i} e_i \frac{f}{f_i} f_i',$$

可知 $u = \gcd(f, f') = \prod_{p \nmid e_i} f_i^{e_i - 1} \prod_{p \mid e_i} f_i^{e_i}$, 于是

$$v = \frac{f}{u} = \prod_{p \nmid e_i} f_i,$$

其中乘积中缺少了某些项, 这些项的次数均是 p 的倍数. 我们注意到多项式 u 中含有我们需要的这些幂次, 且 $e_i \leq n$, 则有如下关系:

$$w = u / \gcd(u, v^n) = \prod_{p \mid e_i} f_i^{e_i},$$

从而 w 为 p 次幂. 由此我们可以得到如下算法:

算法9.10 (有限域无平方部分算法).

输入: 有限域 $\mathbb{F}_q[x]$ 上首一非平凡多项式 f, $\deg f = n$.

输出: f 的无平方部分 g.

1. $u = \gcd(f, f')$, $v = f/u$, $w = u / \gcd(u, v^n)$,
2. 若 $w = 1$, 则输出 v, 否则递归调用本算法计算 $w^{1/p}$ 的无平方部分 v_1.
3. 输出 vv_1.

例9.5. 求 $f = a^2 b^3 c^6 d^9 \in \mathbb{F}_3[x]$ 的无平方部分, 其中 a, b, c, d 是互素且不可约首一多项式.

解: 顺次计算得:

$$\begin{aligned} f' &= 2aa'b^3c^6d^9, \\ u &= \gcd(f, f') = ab^3c^6d^9, \\ v &= f/u = a, \\ w &= u/\gcd(u, v^n) = u/a = b^3c^6d^9, \end{aligned}$$

递归调用算法, 计算得

$$\begin{aligned} f_1 &= w^{1/3} = bc^2d^3, \\ f_1' &= b'c^2d^3 + 2bcc'd^3, \\ u_1 &= cd^3, \\ v_1 &= bc, \\ w_1 &= d^3, \end{aligned}$$

再次递归调用的结果为 $v_2 = d$. 故最后输出 $vv_1v_2 = abcd$. ◇

上节中的例 9.4 给了我们对于算法 9.9 的理解: v_i 序列包含了要处理的无平方部分, $g = v_1 = g_1 \cdots g_m$, $w_1 = \sum_{i=1}^m e_i g_i' g/g_i$, e_i 是 g_i 在 f 中的次数(在下面的说明中 $e_i = i$), 每处理一次, v_i 中去掉次数 i 的项, 如果是在有限域中, 该算法就只能在 $m < p$ 时正确, 因为它会将模 p 相同的次数 i 归于同一个次数 $i \bmod p$ 上, 即有下面的结果:

$$\begin{aligned} h_i &= \prod_{j \equiv i \pmod p} g_j \quad (1 \leq i < p), \\ h_i &= 1 \quad (i \geq p). \end{aligned}$$

假设 $m > p$, $f = \prod_{i=1}^m g_i^i$, 则算出 $h_1, h_2, \ldots, h_{p-1}$ 后, 令 $f_1 = fh_1^{-1}h_2^{-2} \cdots h_{p-1}^{-p+1}$, 则

$$f_1 = \prod_{i \geq p} g_i^{i - i \bmod p},$$

于是

$$f_1^{\frac{1}{p}} = \prod_{i \geq p} g_i^{\left[\frac{i}{p}\right]}.$$

如果我们能够构造递归算法以得到 $f_1^{1/p}$ 的无平方分解 $(s_1, \ldots, s_l)$, 则显然有

$$s_j = \prod_{[\frac{i}{p}] = j} g_i,$$

于是 $\gcd(h_i, s_j) = g_{jp+i}$.

通过前面的讨论, 我们得到了如下一种利用递归进行分解的算法.

算法9.11 (有限域无平方分解).

输入: 有限域 $\mathbb{F}_q$ 上首一不可约多项式 f

输出:f 的无平方分解 $(g_1, g_2, \ldots, g_m)$.

1. 调用算法 9.9 计算出 $h_k (1 \le k < p)$, $f_1 = \dfrac{f}{h_1 h_2^2 \cdots h_k^k}$. 若 $f_1 = 1$ 则输出 $(h_1, \ldots, h_k)$,
2. 递归调用本算法得到 $f_1^{1/p}$ 的无平方分解 $(s_1, s_2, \ldots, s_l)$,
3. 令 $h_{k+1}, \ldots, h_{p-1} = 1$,
4. $g_{jp+i} = \gcd(h_i, s_j) \quad (1 \le i < p, 1 \le j \le l)$,
5. $g_{jp} = \dfrac{s_j}{g_{jp+1} g_{jp+2} \cdots g_{(j+1)p-1}} \quad (1 \le j \le l)$,
6. $g_i = \dfrac{h_i}{g_{p+i} g_{2p+i} \cdots g_{lp+i}} \quad (1 \le i < p)$,
7. 令 m 为最大的使 $g_m \ne 1$ 的下标, 输出 $(g_1, \ldots, g_m)$.

9.5 Berlekamp 算法

最早的有限域上多项式因子分解算法是 Berlekamp 于 1967, 1970 年提出的 [27][28]. 为了引入这个算法, 我们先做一些代数上的准备.

9.5.1 Frobenius 映射和 Berlekamp 子代数

定义9.4. $\mathbb{F}_{q^n}$ 或 $\mathbb{F}_q[x]$ 上(q 为一素数幂)的映射 $\sigma : a \mapsto a^q$ 称为 Frobenius 映射.

注136. 注意到 Frobenius 映射 σ 是线性映射, 且保持加法和乘法.

以下设 $f \in \mathbb{F}_q[x]$ 是一首一无平方因子多项式, $\deg f = n$ 且 $f = \prod_{i=1}^{r} f_i$, f_i 为两两互素且不可约首一多项式. 由中国剩余定理有下面的环同构:

$$R = \mathbb{F}_q[x]/\langle f \rangle \cong \mathbb{F}_q[x]/\langle f_1 \rangle \times \cdots \times \mathbb{F}_q[x]/\langle f_r \rangle = R_1 \times \cdots \times R_r,$$

通过同构映射:

$$\chi : a \bmod f \in R \mapsto (a \bmod f_1, \ldots, a \bmod f_r).$$

定理9.5. *σ 是 R 的自同构.*

证明. 由于 χ 是 R 到 $\prod_{i=1}^{r} R_i$ 的同构, 故可由 σ 在 R_i 上诱导出相应的线性映射 $\sigma_i : a_i \mapsto a_i^q (a_i \in R_i)$, 于是 $\ker\sigma \cong \prod_{i=1}^{r} \ker\sigma_i$. 又由于 R_i 为域, 那么 $\sigma_i(a_i) = a_i^q = 0$ 仅有 q 重根 0, 即 $\dim\ker\sigma_i = 0$, 因此 $\dim\ker\sigma = 0$, σ 是单射. 由于有限维线性空间上的线性变换若是单射则必为同构, 可知 σ 是 R 的自同构. □

定义9.5. 记 $\mathcal{B} = \ker(\sigma - \mathrm{id})$, 其是 $\mathbb{F}_q$ 上 R 的子代数, 也被称为 Berlekamp 子代数.

定理9.6. $\dim\mathcal{B} = r$.

证明. 由于 χ 是 R 到 $\bigotimes_{i=1}^{r} R_i$ 上的同构, 因此我们实际上将两者等同看待, 则 σ 等同于

$$(a \bmod f_1, \ldots, a \bmod f_r) \mapsto (a^q \bmod f_1, \ldots, a^q \bmod f_r).$$

$\forall a \in \mathcal{B}$, 记 $a_i = a \bmod f_i$, 则 $a^q = a \Rightarrow a_i^q = a_i$. 但是 $a_i \in R_i = \mathbb{F}_q[x]/\langle f_i\rangle$, R_i 实际上是一个域, 其上的代数方程 $x^q - x = 0$ 至多有 q 个根, 全体根恰好组成 $\mathbb{F}_q$ 这个子域, 即有 $a_i \in \mathbb{F}_q$. 于是 $a \in \bigotimes_{i=1}^{r} \mathbb{F}_q \Rightarrow \mathcal{B} \subset \bigotimes_{i=1}^{r} \mathbb{F}_q$. 而显然后者也包含于前者, 于是有两者相等, 从而 $\dim\mathcal{B} = r$. □

定义9.6. Frobenius 映射在 R 上自然基 $\{1, x, \ldots, x^{n-1}\}$ 下的表示矩阵记作 $Q \in \mathbb{F}_q^{n\times n}$, 称为 Petr-Berlekamp 矩阵.

推论9.2. *$\mathbb{F}_q$ 上无平方因子 n 次多项式 f 不可约当且仅当 $\mathrm{rank}(Q - I) = n - 1$.*

推论9.3. *$\mathbb{F}_q$ 上无平方因子 n 次多项式 f 的不可约因子的个数为 $\ker\mathcal{B} = n - \mathrm{rank}(Q - I)$.*

取 $\mathcal{B}$ 的一组基 $\{b_1, b_2, \ldots, b_r\}$, 因为 $\mathcal{B} = \bigotimes_{i=1}^{r} \mathbb{F}_q$, 可设这组基在 $R_1 \times R_2 \times \cdots \times R_r$ 中的表示矩阵为

$$B = (b_1, \ldots, b_r) = \begin{pmatrix} b_{11} & b_{21} & \cdots & b_{r1} \\ \vdots & \vdots & & \vdots \\ b_{1r} & b_{2r} & \cdots & b_{rr} \end{pmatrix} \in \mathbb{F}_q^{n\times n}.$$

此为一可逆矩阵, 于是 $\forall 1 \le i < j \le r$, $\exists k (1 \le k \le r)$ 使得 $b_{ik} \ne b_{jk}$, 即 $b_k \bmod f_i \ne b_k \bmod f_j$, 则对于 $b_k - b_{ik}$ 有

$$f_i \mid (b_k - b_{ik}), \quad f_j \nmid (b_k - b_{ik}).$$

则 $b_i - b_{ik}$ 是可以分离 f 的一个多项式.

推论9.4. 任取 $\mathcal{B}$ 的一个基矢 b_k, 任取 $a \in \mathbb{F}_q$, 则 $\gcd(b_k - a, f)$ 可能给出 f 的一个非平凡因子.

9.5.2 Berlekamp 算法的实现

有了上面的准备工作, 下面我们可以来引入 Berlekamp 算法了. 由推论 9.4 可以得到如下算法.

算法9.12 (Berlekamp 算法 1).

输入: $\mathbb{F}_q$ 上无平方因子首一 n 次非平凡多项式 f,

输出: f 可能的非平凡因子, 或者失败.

1. 构作环 $R = \mathbb{F}_q[x]/\langle f\rangle$ 上的 Frobenius 映射的表示矩阵 Q, 即 Petr-Berlekamp 矩阵,
2. 对 $Q - I$ 进行高斯消元法, 求出 $\mathcal{B} = \ker(Q - I)$ 的 r 个基矢 $\{b_1, \ldots, b_r\}$,
3. 随机任取一个基矢 $b_k (1 \leq k \leq r)$, 任取 $a \in \mathbb{F}_q$, 对 U 中任何一个元素 u, 计算 $v = \gcd(b_k - a, u)$, 若 $v \neq 1$ 且 $v \neq u$, 则输出 v, 否则输出失败.

注137. 求 Petr-Berlekamp 矩阵时可先用快速求幂算法算出 $x^q \bmod f$, 进而求出 $x^{qi} (1 \leq i \leq n)$.

下面是另外一种概率性的 Berlekamp 算法 [174], 能给出 f 的可能因子.

算法9.13 (Berlekamp 算法 2).

输入:$\mathbb{F}_q$ 上无平方因子首一 n 次非平凡多项式 f,

输出:f 的可能非平凡因子, 或者失败.

1. 构作环 $R = \mathbb{F}_q[x]/\langle f\rangle$ 上的 Petr-Berlekamp 矩阵 Q,
2. 对 $Q - I$ 进行高斯消元法, 求出 $\mathcal{B} = \ker(Q - I)$ 的 r 个基矢 $b_1, \ldots, b_r$,
3. 独立地随机选取 $c_1, c_2, \ldots, c_r \in \mathbb{F}_q$, 计算 $a = \sum_{i=1}^{r} c_i b_i$,
4. $g_1 = \gcd(a, f)$, 若 $g_1 \neq 1$ 且 $g_1 \neq f$ 则输出 g_1,
5. $b = a^{(q-1)/2} \bmod f$, $g_2 = \gcd(b - 1, f)$,
6. 若 $g_2 \neq 1$ 且 $g_2 \neq f$ 则输出 g_2, 否则输出失败.

该算法的正确性证明与奇素数幂同次因子分解(算法 9.2)类似, 只须注意到 $\chi_i(a) \in \mathbb{F}_q$, 这样我们有 $a^{(q-1)/2} = 0$ 或 1, 两种取值等概率为 1/2.

为了引入特征为 2 的域上与算法 9.13 对应的 Berlekamp 算法, 我们证明如下引理.

引理9.5. *设 a 是 $\mathcal{B} = \ker(Q - I)$ 中任意一随机元素, $b = T_k(a)$ 为迹多项式, 则 $\gcd(b-1, f)$ 可能给出 f 的一个非平凡因子, 且失败概率不超过 1/2.*

证明. 首先, 由于 $\chi_i(a) \in \mathbb{F}_{2^k} = \mathbb{F}_q$, 有 $\chi_i(T_k(a)) = T_k(\chi_i(a)) = 0$ 或 1, 且两值等概率. 当且仅当 $\chi_i(b)$ 全为 0 或 1 时, $b = 0$ 或 1, 此概率为 $2^{1-r} \leq 1/2$. 易知, 当且仅当 $b \notin \mathbb{F}_2$ 时, $\gcd(b-1, f)$ 含有 f 的非平凡因子. □

于是对于特征为 2 的域 $\mathbb{F}_q = \mathbb{F}_{2^k}$, 有如下的:

算法9.14 (Berlekamp 算法 3)**.**

将算法 9.13 中第 5 步改为计算 $b = T_k(a)$, 其余不变.

9.6 各算法复杂度比较

在前面几节, 我们着重介绍了不同次因子分解算法, 同次因子分解算法(Cantor-Zassenhaus 算法)以及 Berlekamp 算法, 并且也提到了 von zur Gathen 和 Shoup 的 Frobenius 迭代算法, Kaltofen 和 Shoup 的 Baby Step-Giant Step 算法. 这些算法渐近复杂度可用图9.1表示(参见 [174]14.8 节).

由图9.1我们可以看出, 在有限域的阶较小时, 即 q 比起 n 较小时, 较优的算法是 Kaltofen 和 Shoup 的算法, 然后是 Frobenius 迭代算法, 在 q 很大时, 各算法的渐近复杂度是一样的.

9.7 不可约性检测和不可约多项式的构造

若要检测一个多项式的不可约性, 前面的因子分解的方法当然也是适用的, 只需修改相应的算法终止条件即可, 下面再介绍一个比较简单的检测方法 [174].

推论9.5. *n 次非平凡多项式 $f \in \mathbb{F}_q[x]$ 是不可约的当且仅当:*

1. $f \mid x^{q^n} - x$,
2. *对满足 $t \mid n$ 的素数 t, 都有* $\gcd(x^{q^{n/t}} - x, f) = 1$.

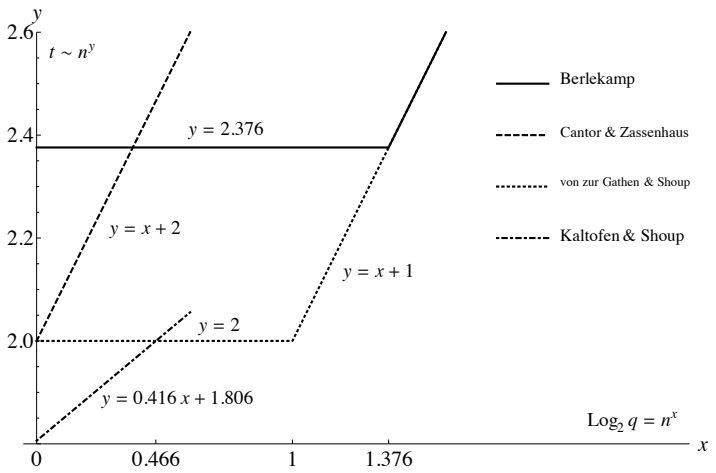

图 9.1: 各种算法渐近复杂度比较

证明. 由定理 9.2 知道上面两个条件是必要的. 再设两个条件满足, 则首先由条件 1 及定理 9.2 知 f 的不可约因子次数均整除 n, 不妨设有这样一个非平凡因子 g 且 $\deg g = d < n$, 则存在 n 的素因子 t 使 $d \mid (n/t)$, 于是 $g \mid \gcd(x^{q^{n/t}} - x, f)$, 矛盾, 于是 f 不可约. □

算法9.15 (有限域上不可约性检测).

输入: n 次多项式 $f \in \mathbb{F}_q[x]$,

输出: 不可约或可约.

1. 调用快速求幂算法计算 $x^q \bmod f$,
2. 调用模复合算法 9.16 计算 $a = x^{q^n} \bmod f$, 若 $a \neq x$ 则输出可约,
3. 对于所有 n 的素因子 t, 调用模复合算法计算 $a = x^{q^{n/t}} \bmod f$, 若 $\gcd(b - x, f) \neq 1$ 则输出可约,
4. 输出不可约.

注138. 模复合(Modular composition)算法 [174] 是快速矩阵乘法的应用, 这里不加证明地给出如下.

算法9.16 (模复合算法).

输入: 设 R 为环, f, g, $h \in R[x]$, 且 $\deg g$, $\deg h < \deg f = n$, f 首一且不为零,

输出: $g(h) \bmod f \in R[x]$.

1. $m = [n^{1/2}]$, 并设 $g = \sum_{0 \le i < m} g_i x^{mi}$, 其中 $g_0, \ldots, g_{m-1} \in R[x]$ 的次数少于 m,
2. 对于 $2 \le i \le m$, 计算 $h^i \bmod f$,
3. 令 $A \in R^{m \times n}$, 其行由 1, $h \bmod f$, $\cdots$, $h^{m-1} \bmod f$ 的系数组成, $B \in R^{m \times m}$, 其行由 $g_0, \ldots, g_{m-1}$ 的系数组成, 计算 $BA \in R^{m \times n}$,
4. 对于 $0 \le i < m$ 循环, 令 r_i 为 BA 第 i 行作为系数构成的多项式, 并利用 Horner 规则计算 $b = \sum_{0 \le i < m} r_i (h^m)^i \bmod f$,
5. 输出 b.

构造一个不可约多项式的最基本的想法就是随机取一个多项式, 再对其作不可约性检测. 于是我们必须要对随机选取取到不可约多项式的概率进行估计.

引理9.6. 设 q 是一素数幂, n 是正整数, 则 $\mathbb{F}_q[x]$ 中 n 次首一不可约多项式的个数 $I(n,q)$ 满足

$$\frac{q^n - 2q^{n/2}}{n} \le I(n,q) \le \frac{q^n}{n},$$

因此随机选取 $\mathbb{F}_q[x]$ 中 n 次首一多项式为不可约的概率 p_n 满足

$$\frac{1}{n}(1 - \frac{2}{q^{n/2}}) \le p_n \le \frac{1}{n}.$$

证明. 令 f_n 为 $\mathbb{F}_q[x]$ 中所有首一不可约 n 次多项式的乘积, 则 $\deg f_n = n \cdot I(n,q)$, 由定理 9.2 知

$$x^{q^n} - x = \prod_{d|n} f_d = f_n \cdot \prod_{d|n, d<n} f_d.$$

对上式取次数, 有

$$q^n = \deg f_n + \sum_{d|n, d<n} \deg f_d,$$

因此

$$q^n \ge \deg f_n = n \cdot I(n,q) \Rightarrow I(n,q) \le \frac{q^n}{n}.$$

另外,

$$\sum_{d|n,d<n} \deg f_d \le \sum_{1\le d\le n/2} \deg f_d \le \sum_{1\le d\le n/2} q^d < \frac{q^{n/2+1}-1}{q-1} \le 2q^{n/2},$$

因此

$$n\cdot I(n,q) = \deg f_n = q^n - \sum_{d|n,d<n} \deg f_d \ge q^n - 2q^{n/2},$$

由此可得到关于 $I(n,q)$ 下界的估计.

由于 n 次首一多项式共有 q^n 个, 因此不可约的概率为 $p_n = I(n,q)/q^n$, 由此得到关于概率的估计. □

注139. 由于 $q^n = \sum_{d|n} \deg f_d$, 则由 Mobius 反演公式可给出 $I(n,q)$ 的精确表达(参见 [10]26–29 页)

$$nI(n,q) = \deg f_n = \sum_{d|n} \mu\left(\frac{n}{d}\right) q^d.$$

由引理 9.6 我们看到当 n 很大时, n 次多项式不可约的概率趋向于 $1/n$. 因此在 n 不是很大时, 可以用下面的算法来产生不可约多项式.

算法9.17 (产生不可约多项式算法).

输入: 素数幂 q 和正整数 n,

输出: 一个随机生成的 n 次首一多项式 $f \in \mathbb{F}_q[x]$.

1. 随机生成一个首一 n 次多项式 $f \in \mathbb{F}_q[x]$,
2. 对于 $i = 1, \ldots, [n/2]$ 循环, 令 $g_i = \gcd(x^{q^i} - x, f)$, 若 $g_i \ne 1$ 则转 1,
3. 输出 f.

10 整系数多项式因子分解

下面我们讨论 $\mathbb{Z}$ 和 $\mathbb{Q}$ 上的多项式的因子分解. 由高等代数学知识, 我们知道对于 $\mathbb{Z}[x]$ 上的多项式 f, 其在 $\mathbb{Q}[x]$ 中的不可约因子分解可对应于在 $\mathbb{Z}[x]$ 中的不可约因子分解, 设 f 是本原多项式, 其在 $\mathbb{Q}[x]$ 中的不可约因子分解为若

$$f = f_1 f_2 \cdots f_r (f_i \in \mathbb{Q}[x]),$$

则有

$$f = f_1' f_2' \cdots f_r' (f_i' \in \mathbb{Z}[x]).$$

其中 f_i' 可由 f_i 乘以其各个系数既约分母的最小公倍数再本原化得到. 于是 $\mathbb{Z}[x]$ 上任一多项式 f 的因子分解归结于 $\mathbb{Z}$ 上的因子分解和 $\mathbb{Z}[x]$ 上本原多项式的因子分解.

很自然的, 由前面的最大公因子模方法我们可以想到用模方法来求解因子分解问题. 我们可以利用有限域上无平方因子分解的方法得到 $\mathbb{Z}[x]$ 上的无平方因子本原多项式 f, 这时, 我们会遇到如下一些问题:

1. 素数 p 的选取要足够大, 以使我们能从 $f \bmod p$ 得到 f, 这一点我们可由模公因子算法中介绍的 Mignotte 界理论得到.

2. 虽然 f 已是无平方因子, 但 $f \bmod p$ 却不一定是无平方因子的. 如多项式 $f = x^2 + 5x + 4$ 无平方因子, 但 $f \bmod 3 = x^2 + 2x + 1 = (x+1)^2$. 那么在随机选取素数 p 的时候如何使得 $f \bmod p$ 也是无平方因子呢? 这一点依赖于结式理论并且将在后文中回答.

3. 当我们在 $\mathbb{F}_p[x]$ 中将多项式分解后, 因为若 f 有不可约分解 $f = f_1 f_2 \cdots f_r$, 则 $\overline{f} = \overline{f_1} \cdots \overline{f_r}$, 但 $\overline{f_i}$ 不一定不可约. 因此还要考虑如何将 $f \bmod p$ 的分解对应到 f 的分解. 最简单的方法是尝试每一种可能的因子组合. 当然, 用这种尝试的方法, 其复杂度是指数阶的.

综上, 我们首先要进入 $\mathbb{F}_p[x]$ 中求得因子分解, 这一步我们可以利用"大素数"方法和"小素数"方法. . 第二步是由 $\mathbb{F}_p[x]$ 返回 $\mathbb{Z}[x]$ 中, 求得最终结果, 可以用因子组合的方法. 后面还会介绍一种多项式复杂度的格中短向量方法, 尽管其理论上十分优美, 但是没有因子组合算法实用.

10.1 大素数模方法和因子组合算法

利用结式理论, 我们先讨论 $f \bmod p$ 在什么情况下是无平方因子的.

定义10.1. 对于域 F 上的多项式 f, 定义其判别式为 $\operatorname{disc}(f) = \operatorname{res}(f, f')$.

由推论 9.1 知, $\overline{f} = f \bmod p$ 是无平方因子的当且仅当 $\operatorname{disc}(\overline{f}) \neq 0$.

引理10.1. *设 $'$ 表示形式微商, 则有 $\overline{f'} = \overline{f}'$.*

定理10.1. *令 $f \in \mathbb{Z}[x]$ 是一非零无平方因子多项式, p 为素数且 $p \nmid \operatorname{lc}(f)$, 则 $\overline{f}$ 是无平方因子的当且仅当 $p \nmid \operatorname{disc}(f)$.*

证明. $\overline{f}$ 无平方因子 $\Leftrightarrow \operatorname{disc}(\overline{f}) \neq 0 \Leftrightarrow \operatorname{res}(\overline{f}, \overline{f}') \neq 0$, 再由上面引理知:

$$\operatorname{res}(\overline{f}, \overline{f}') \neq 0 \Leftrightarrow \operatorname{res}(\overline{f}, \overline{f'}) \neq 0.$$

又 $p \nmid \operatorname{lc}(f)$, 则由引理 8.2 知

$$\operatorname{res}(\overline{f}, \overline{f}') \neq 0 \Leftrightarrow \overline{\operatorname{res}(f, f')} \neq 0 \Leftrightarrow p \nmid \operatorname{disc}(f).$$

证毕. □

由于 $\operatorname{lc}(f) \mid \operatorname{res}(f, f')$, 我们有下面的:

推论10.1. *f 是 $\mathbb{Z}[x]$ 上非零无平方因子多项式, 则 $\overline{f}$ 无平方因子当且仅当 $p \nmid \operatorname{disc}(f) = \operatorname{res}(f, f') \in \mathbb{Z} \setminus \{0\}$.*

注140. 由推论 10.1 可知, p 可以随机选取, 只有很小概率会使得 $\overline{f}$ 是有重因子的. 实际检测时, 我们不需要计算结式以判别选择的 p 是否可行, 只需要直接计算 $\gcd(\overline{f}, \overline{f}')$ 即可.

设 $f \in \mathbb{Z}[x]$ 为本原多项式, 有分解 $f = f_1 f_2 \cdots f_k$, 且在模 p 下有:

$$\overline{f} = \overline{f_1 f_2} \cdots \overline{f_k} = \overline{\mathrm{lc}(f)} g_1 g_2 \cdots g_r,$$

其中 g_i 为 $\mathbb{F}_p[x]$ 上首一不可约多项式. 若 $p/2$ 比 Mignotte 界 $(n+1)^{1/2} 2^n |\mathrm{lc}(f)| \cdot \|f\|_\infty$ 小, 则有下面的等式:

$$\frac{\mathrm{lc}(f)}{\mathrm{lc}(f_1)} f_1 = \mathrm{lc}(f) \prod_{i \in S} g_i \in \mathrm{Z}[x],$$

该式原先在 $\mathbb{F}_p[x]$ 上就已成立了. 其中指标集 S 为 $S = \{i \in \{1, \ldots, r\} \big| \ g_i \mid \overline{f_1}\}$. 由此启发我们得到算法 10.1 .

算法10.1 (整系数多项式因子分解 1:大素数和因子组合算法).

输入: 无平方因子 n 次本原多项式 $f \in \mathbb{Z}[x]$, 其中 $\mathrm{lc}(f) > 0$ 且 $\|f\|_\infty = A$,

输出: f 在 $\mathbb{Z}[x]$ 上的不可约因子 $\{f_1, \ldots, f_k\}$.

1. 若 $n = 1$ 则输出 f, 否则 $b = \mathrm{lc}(f), B = (n+1)^{1/2} 2^n A b$,
2. 随机任取一个奇素数 $p \in (2B, 4B)$, 直至 $\gcd(\overline{f}, \overline{f}') = 1 \in \mathbb{F}_p[x]$, 即满足推论 10.1 条件,
3. 利用有限域上因子分解算法求出 $g_1, \ldots, g_r \in \mathbb{Z}[x]$, 其无穷范数均比 $p/2$ 要小, 且在 $\mathbb{F}_p$ 上不可约, 于是 $f \equiv b g_1 \cdots g_r \pmod{p}$,
4. $T = \{1, \ldots, r\}$, $s = 1$, $G = \varnothing$, $f^* = f$, (此步之后即为因子组合)
5. 当 $2s \leq \#T$ 时循环执行下面 4 步, 否则转第 10 步,
6. 枚举 T 的所有 s 元子集 S, 并做下两步 7, 8 循环:
7. 计算 $g^*, h^* \in \mathbb{Z}[x]$ 使得其无穷范数比 $p/2$ 要小并且 $g^* \equiv b \prod_{i \in S} g_i \pmod{p}$, $h^* \equiv b \prod_{i \in T \setminus S} g_i \pmod{p}$,
8. 若 $\|g^*\|_1 \|h^*\|_1 \leq B$ 则 $T = T \setminus S$, $G = G \cup \{\mathrm{pp}(g^*)\}$, $f^* = \mathrm{pp}(h^*)$, $b = \mathrm{lc}(f^*)$, 跳出 6, 7, 8 循环并转第 5 步,
9. $s = s + 1$,
10. 输出 $G \cup \{f^*\}$.

算法有效性. 由第 2 步 $p > B \Rightarrow p \nmid b$ 我们已经知道 $\overline{f}$ 是无平方因子的. 在第 8 步中, 若条件真则有 $g^*h^* = bf^*$, 因为由 $g^*h^* \equiv bf^* \pmod p$ 和 $\|g^*h^*\|_\infty \leq \|g^*h^*\|_1 \leq \|g^*\|_1\|h^*\|_1 \leq B < p/2$ 知等式是成立的. 记 f 的因子 $u \in \mathbb{Z}[x]$, 其在 $\mathbb{F}_p[x]$ 中不可约因子个数为 $\mu(u)$. 现在我们要归纳证明在每次到第 5 步时, 有下面命题成立:

1. $f^* \equiv b\prod_{i\in T} g_i \pmod p$, $\quad b = \mathrm{lc}(f^*)$, $\quad f = f^*\prod_{g\in G} g$,
2. G 中多项式均不可约,
3. f^* 本原且它的任何一个不可约因子 $u \in \mathbb{Z}[x]$ 有 $\mu(u) \geq s$.

初始时命题显然成立, 假设命题在每次循环进行到第 7 步前均是成立的, 此时经过第 7 步后当第 8 步的条件成立时, 各量均要发生变化, 根据前面的分析则有 $g^*h^* = bf^*$, 于是 $\mathrm{pp}(g^*)$ 是 $\mathrm{pp}(bf^*) = f^*$ 的因子. 由于 $\mu(g^*) = s$ 且对任何 f^* 的不可约因子 u 有 $\mu(u) \geq s$, 则 $\mathrm{pp}(g^*)$ 是 f^* 的不可约因子. 当 f^* 有一个不可约因子 g 满足 $\mu(g) = s$ 时, 当循环到 s 时必然能将此因子选出, 这一点可以构造来证明, 即取指标集 S 为 g 在 $\mathbb{F}_p[x]$ 中不可约因子的编号.

最后一步是证明在第 5 步时, 若 $2s > \#T$, 则 f^* 是不可约的. 令 $g \in \mathbb{Z}[x]$ 是 f^* 的一个不可约因子且 $h = f^*/g$ 非平凡, 于是 $s \leq \mu(g), \mu(h) \leq \#T$. 但是 $\mu(g) + \mu(h) = \#T$, 且 $s > \#T/2$, 则 h 必为常数, f^* 必不可约. □

例10.1. 求 $f = 4x^4 + 13x^3 + 28x^2 + 27x + 18$ 在 $\mathbb{Z}[x]$ 上的分解.

解: f 是本原的, 且 $f' = 16x^3 + 39x^2 + 56x + 27$, $\mathrm{disc}(f) = 1656288$, 于是 f 无平方因子. 此时 $n = 4$, $A = 28$, $b = \mathrm{lc}(f) = 4$, 则 $B = (n+1)^{1/2}2^nAb = 1792\sqrt{5} = 4007.03$, 取素数 $p = 8017 > 2B$ 且 $p \nmid \mathrm{disc}(f)$, 此时可以得到 $\mathbb{F}_{8017}$ 上的分解:

$$f \equiv 4(x-955)(x+957)(x^2-2003x-4007) \pmod p.$$

首先 $s = 1$, 若取 $S = \{1\}$, 则

$$g^* \equiv 4(x-955) \equiv 4x - 3820 \pmod p,$$

$$h^* \equiv 4(x+957)(x^2-2003x-4007) \equiv 4x^3 + 3833x^2 - 3226x - 2275 \pmod p,$$

$$\|g^*\|_1\|h^*\|_1 = (4+3820)(4+2833+3226+2275) > B,$$

同样的取 $S = \{2\}$ 时也可验证是不可行的. 若取 $S = \{3\}$, 则

$$g^* \equiv 4(x^2-2003x-4007) \equiv 4x^2 + 5x + 6 \pmod p,$$

$$h^* \equiv 4(x-955)(x+957) \equiv 4x^2+8x+12 \pmod{p},$$

此时 $\|g^*\|_1\|h^*\|_1 = 15*24 = 360 < B$, 则 $G = \{4x^2+5x+6\}$, $f^* = \mathrm{pp}(h^*) = x^2+2x+3$, $b = \mathrm{lc}(f^*) = 1$, $T = \{1,2\}$. 下一步 $s = 2$, 循环条件不满足, 则 $G = G\bigcup\{f^*\} = \{x^2+2x+3, 4x^2+5x+6\}$. ◇

利用算法 10.1 我们可以得到如下对于一般的多项式的分解算法.

算法10.2 (整系数多项式的因子分解 2).

输入: $f \in \mathbb{Z}[x]$, $\deg f = n > 1$ 且 $\|f\|_\infty = A$,

输出: 常数 $c \in \mathbb{Z}$ 和序对集 $\{(f_1,e_1),\ldots,(f_k,e_k)\}$, 其中 $f_i \in \mathbb{Z}[x]$ 均是不可约两两互素的多项式, $e_i \in \mathbb{N}$, 且 $f = c\prod_{i=1}^k f_i^{e_i}$.

1. $c = \mathrm{cont}(f)$, $g = \mathrm{pp}(f)$, 若 $\mathrm{lc}(f) < 0$ 则 $c = -c$, $g = -g$,
2. 调用算法 9.9 得到分解 $g = \prod_{1\le i\le s} g_i^i$, 且 $\mathrm{lc}(g_i) > 0$, g_s 非平凡,
3. $G = \varnothing$,
4. 对 $1 \le i \le s$ 循环, 调用上面算法 10.1 得到 g_i 的所有不可约因子 $h_1,\ldots,h_t \in \mathbb{Z}[x]$, $G = G\bigcup\{(h_1,i),\ldots,(h_t,i)\}$,
5. 输出 c 和 G.

注141. 当然, 在算法第 4 步分解无平方因子多项式时, 也可以调用后面几节将要介绍的 Hensel 提升或格中短向量算法进行因子分解.

10.2 Hensel 提升理论

首先我们来简单介绍一下为什么要用 Hensel 提升算法以及 Hensel 提升大致要解决的是什么问题.

依照我们前面介绍的求整系数多项式最大公因子的小素数模算法的思想, 如果要对本原多项式 $f \in \mathbb{Z}[x]$ 进行因子分解, 应当选取一系列小素数 $p_1,p_2,\ldots,p_k$ 并在相应的环 $\mathbb{F}_{p_i}[x]$ 中作分解

$$f \equiv b_i h_{1i} h_{2i} \cdots h_{r_i i} \pmod{p_i},$$

其中 h_{ji} 均为首一不可约多项式, b_i 是领项系数, r_i 是 f 在 $\mathbb{F}_{p_i}[x]$ 中分解得到的不可约多项式个数. 然后利用中国剩余定理, 将各个环中的分解合并, 设

$m = \prod_{1\le i\le k} p_i$, 我们将得到

$$f \equiv bh_1h_2\cdots h_r \pmod m,$$

只要 m 依照 Mignotte 界选取, 那么由此进行因子组合算法可还原到得 $\mathbb{Z}[x]$ 中的分解. 这样的设想虽然很好, 但是会存在如下问题:

- 在不同的环 $\mathbb{F}_{p_i}[x]$ 中分解得到的不可约因子的个数未必相同, 即 r_i 未必全相等.
- 即使各个 r_i 均是相等的, 但是不同环中得到的不可约因子不能一一对应起来, 这会给中国剩余定理算法带来麻烦.例如考虑多项式 $f = (x+3)(x+5)$ 的分解, 在 $\mathbb{F}_3[x]$ 和 $\mathbb{F}_5[x]$ 中有

  $$f \equiv x(x+2) \pmod 3, \quad f \equiv x(x+3) \pmod 5,$$

  虽然分解得到相同的因子 x, 而实际上两个分解中的 x 并不是对应的.

基于以上考虑, 我们需要换一个思路. 假设对于某个小素数 p, 我们已有分解

$$f \equiv bh_1h_2\cdots h_r \pmod p,$$

如果能通过某种方法将其“提升”, 从而得到分解

$$f \equiv ag_1g_2\cdots g_r \pmod{p^l},$$

其中 $a \equiv b \pmod p$, $g_i \equiv h_i \pmod p$, 而 p^l 已足够大, 那么亦能达到同样的目的. Hensel[83] 于 1918 年提出了提升算法, 用以解决这样的问题.

10.2.1 Hensel 单步算法

当我们已知一个分解 $f \equiv gh \pmod p$(g, h 互素)时, 最简单的问题是如何获得分解 $f \equiv g^*h^* \pmod{p^2}$, 即将其“提升”. 由于 p 是素数, 则存在 s, $t \in \mathbb{Z}[x]$ 使得 $sg + th \equiv 1 \pmod p$, 如果我们取:

$$e = f - gh, \quad g^* = g + te, \quad h^* = h + se,$$

则有

$$\begin{aligned} f - g^*h^* &= f - (g+te)(h+se) = f - gh - (sg+th)e - ste^2 \\ &= (1 - sg - th)e - ste^2 \equiv 0 \pmod{p^2}, \end{aligned}$$

这样可以达到我们的要求, 下面是一个具体计算的例子.

例10.2. 考虑 $f = x^4 - 1$, $m = 5$, $h = x - 2$, $g = x^3 + 2x^2 - x - 2$, $s = -2$, $t = 2x^2 - 2x - 1$ 的情况.

解: 顺次计算有

$$e = f - gh = 5x^2 - 5,$$

$$g^* = g + te = 10x^4 - 9x^3 - 13x^2 + 9x + 3,$$

$$h^* = h + se = -10x^2 + x + 8.$$

我们看到, $\deg g^* > \deg g$, $\deg h^* > \deg h$, 这种规模的增大无疑对后续的提升造成更多的计算负担, 并且次数的提高时我们无法得到正确的因子分解结果. ◇

鉴于提升时 g 和 h 的次数增长, 我们需要对此方法作一些改动, 在提出改动后的 Hensel 单步提升算法之前, 我们先给出如下引理.

引理10.2. *在 $\mathbb{Z}[x]$ 中, 我们有如下结论:*

1. *设 f, $g \in \mathbb{Z}[x]$, 其中 g 非零且首一, 则存在唯一的多项式 q, $r \in \mathbb{Z}[x]$ 使得 $f = qg + r$ 且 $\deg r < \deg g$.*
2. *f, g, q, r 同上, 若对于某个整数 m 有 $f \equiv 0 \pmod{m}$, 则 $q \equiv r \equiv 0 \pmod{m}$.*

证明. 第一条基本同 Euclid 环中的证明方法, 只要注意到 g 是首一的即可. 对于第二条, 由 $f \equiv 0 \pmod{m}$ 知必有 $\mathbb{Z}[x]$ 中的多项式 f^* 使得 $f = mf^*$, 于是必存在唯一的 q^*, r^* 使得

$$f^* = q^*g + r^*, \quad \deg r^* < \deg q^*,$$

此时有 $f = mf^* = (mq^*)g + (mr^*)$, 由唯一性得 $q = mq^*$, $r = mr^*$. □

注142. 将 2 的条件改为 $f \equiv qg + r \pmod{m^2}$, 结论也是正确的. 只需将证明中的 $\mathbb{Z}[x]$ 相应地改为 $\mathbb{Z}_{m^2}[x]$ 即可.

我们可以给出如下的单步 Hensel 提升(Hensel Step)算法.

算法10.3 (单步 Hensel 提升).

输入: 整数 $m \in \mathbb{Z}$, 多项式 $f, g, h, s, t \in \mathbb{Z}[x]$ 使得

$$f \equiv gh \pmod{m}, \quad sg + th \equiv 1 \pmod{m},$$

其中 h 首一, 且 $\deg f = n = \deg g + \deg h$, $\deg s < \deg h$, $\deg t < \deg g$.

输出: 多项式 $g^*, h^*, s^*, t^* \in \mathbb{Z}[x]$ 使得

$$f \equiv g^* h^* \pmod{m^2}, \quad s^* g^* + t^* h^* \equiv 1 \pmod{m^2},$$

h^* 首一, $g^* \equiv g \pmod{m}$, $h^* \equiv h \pmod{m}$, $s^* \equiv s \pmod{m}$, $t^* \equiv t \pmod{m}$, $\deg g^* = \deg g$, $\deg h^* = \deg h$, $\deg s^* < \deg h^*$, $\deg t^* < \deg g^*$.

1. 计算 $e, q, r, g^*, h^* \in \mathbb{Z}[x]$ 使得 $\deg r < \deg h$ 且

$$e \equiv f - gh \pmod{m^2}, \qquad se \equiv qh + r \pmod{m^2},$$
$$g^* \equiv g + te + qg \pmod{m^2}, \qquad h^* \equiv h + r \pmod{m^2},$$

2. 计算 $b, c, d, s^*, t^* \in \mathbb{Z}[x]$ 使得 $\deg d < \deg h^*$ 且

$$b \equiv sg^* + th^* - 1 \pmod{m^2}, \qquad sb \equiv ch^* + d \pmod{m^2},$$
$$s^* \equiv s - d \pmod{m^2}, \qquad t^* \equiv t - tb - cg^* \pmod{m^2},$$

3. 输出 g^*, h^*, s^*, t^*.

算法有效性. 我们验证各项输出的确满足要求. 首先验证 $f \equiv g^* h^* \pmod{m^2}$,

$$\begin{aligned} f - g^* h^* &\equiv f - (g + te + qg)(h + r) \equiv f - (g + te + qg)(h + se - qh) \\ &\equiv (f - gh) - (sg + th)e - ste^2 + q^2 gh + (th - sg)eq + qgh - qgh \\ &\equiv (1 - sg - th)e - ste^2 + q^2 gh + (th - sg)eq \equiv 0 \pmod{m^2}, \end{aligned}$$

这是因为根据注 142 及 $se \equiv qh + r \pmod{m^2}$, 我们由 $se \equiv 0 \pmod{m}$ 得到 $q \equiv r \equiv 0 \pmod{m}$.

由 $\deg r < \deg h$ 知 h^* 也是首一的, 且

$$g^* - g \equiv te + qg \equiv 0 \pmod{m},$$

$$h^* - h \equiv r \equiv 0 \pmod{m},$$

由 $h^* \equiv h + r \pmod{m^2}$ 也可得到 $\deg h^* = \deg h$, 于是 $\deg g^* = \deg f - \deg h^* = \deg f - \deg h = \deg g$.

其次验证 $s^*g^* + t^*h^* \equiv 1 \pmod{m}$,

$$\begin{aligned} s^*g^* + t^*h^* &\equiv (s-d)g^* + (t - tb - cg^*)h^* \\ &\equiv (s - sb + ch^*)g^* + (t - tb - cg^*)h^* \equiv (sg^* + th^*)(1-b) \\ &\equiv (sg^* + th^*)(2 - sg^* - th^*) \equiv 1 - (sg^* + th^* - 1)^2 \equiv 1 - (sg + th - 1)^2 \\ &\equiv 1 \pmod{m^2}, \end{aligned}$$

由 $sb \equiv 0 \pmod{m}$ 可知 $c \equiv d \equiv 0 \pmod{m}$, 于是

$$s^* - s \equiv d \equiv 0 \pmod{m},$$

$$t^* - t \equiv tb + cg^* \equiv 0 \pmod{m},$$

而 $\deg d < \deg h^*$, 于是 $\deg s^* \leq \deg(s-d) < \deg h^*$, 由 $s^*g^* + t^*h^* \equiv 1 \pmod{m^2}$ 知 $\deg t^* < \deg g^*$. 所有结论证毕. □

既然本节开始提出的方法已经能够解决问题, 为什么还要引入上面的算法呢? 我们通过下面一个例子来说明问题:

例10.3. 仍然考虑例 10.2 $f = x^4 - 1$, $m = 5$, $h = x - 2$, $g = x^3 + 2x^2 - x - 2$, $s = -2$, $t = 2x^2 - 2x - 1$ 的情况.

解: 我们用算法 10.3 来进行提升:

e 任取为 $5x^2 - 5$, 则对 se 进行 h 的带余除法有:

$$se = -10x^2 + 1 \equiv (-10x + 5)h - 5 \pmod{25},$$

于是 $q = -10x + 5$, $r = -5$, 故

$$g^* \equiv g + te + qg \equiv x^3 + 7x^2 - x - 7 \pmod{25},$$

$$h^* \equiv h + r \equiv x - 7 \pmod{25},$$

$$b \equiv sg^* + th^* - 1 \equiv -5x^2 - 10x - 5,$$

$$c = 10x - 10, \quad d = -10,$$

$$s^* \equiv s - d \equiv 8 \pmod{25},$$

$$t^* \equiv t - tb - cg^* \equiv -8x^2 - 12x - 1.$$

正如 Hensel 单步提升算法所提到的, 我们有 $\deg g^* = \deg g$, $\deg h^* = \deg h$.

◇

我们可归纳地利用单步 Hensel 算法, 依次对 m, m^2, m^4, $\cdots$ 使用. 由于对任何正整数 l, 我们总可找到比其大的 2 的幂次, 于是有下面的:

定理10.2 (Hensel 引理). *对于给定的正整数 l 以及算法 10.3 中输入的条件, 我们可以将 m^2 用 m^l 代替, 仍然得到满足条件的输出.*

定理10.3 (Hensel 提升的唯一性). *对于非零整数 m 和正整数 l 以及非零多项式 $g, h, g^*, h^*, s, t \in \mathbb{Z}[x]$, 其中 $sg + th \equiv 1 \pmod{m}$, $\mathrm{lc}(g)$ 和 $\mathrm{lc}(h)$ 不是 $\mathbb{Z}_m$ 中的零因子, g 和 g^* 有同样的领项和次数, 且模 m 同余; 对于 h 和 h^* 也有相似的条件. 此时若 $gh \equiv g^*h^* \pmod{m^l}$, 则 $g \equiv g^* \pmod{m^l}$, $h \equiv h^* \pmod{m^l}$.*

证明. 假设结论不成立, 即 $g \not\equiv g^* \pmod{m^l}$ 或 $h \not\equiv h^* \pmod{m^l}$, 不妨假设前者不成立, 于是我们可以找到最大的指标 $i (1 \le i < l)$ 使得 $m^i \mid g - g^*$ 且 $m^i \mid h - h^*$, 不妨设 $g^* - g = um^i$, $h^* - h = vm^i$ 且 $m \nmid u$. 则

$$0 \equiv g^*h^* - gh \equiv g^*(h^* - h) + h(g^* - g) \equiv (g^*v + hu)m^i \pmod{m^l},$$

于是 $m \mid m^{l-i} \mid (g^*v + hu)$, 若 f 将模 m 的象以 $\overline{f}$ 来记, 则有

$$\overline{sg} + \overline{th} = 1, \quad \overline{g^*} = \overline{g}, \quad \overline{g^*v} + \overline{hu} = 0.$$

因此

$$0 = \overline{t}(\overline{g^*v} + \overline{hu}) = \overline{tgv} + (1 - \overline{sg})\overline{u} = (\overline{tv} - \overline{su})\overline{g} + \overline{u},$$

于是 $\overline{g} \mid \overline{u}$, 又 g 和 g^* 的领项和次数均相同, 我们有 $\deg \overline{u} < \deg \overline{g}$, 于是由整除性知 $\overline{u} = 0$, 这与 $m \nmid u$ 矛盾. □

10.2.2 利用因子树进行多因子 Hensel 提升

前面所说的均是二因子的 Hensel 提升, 对于多因子, 情况有些不同. Victor Shoup 提出了一种利用"因子树"进行提升的方法, 最早在 NTL 库中实现(参见 [174]15.5 节), von zur Gathen[172] 于 1984 年提出了同时提升多个因子的算法. 本节我们介绍因子树方法.

我们先给出因子树的定义:

定义10.2 (因子树(Factor Tree)). *对于 $\mathbb{Z}[x]$ 中的多项式 f, 以及正整数 m(例如我们可以取作素数 p), 设有整数 a 使得 $a\mathrm{lc}(f) \equiv 1 \pmod{m}$, 则 f 模 m 的因子树是指一个二叉树 τ:其根结点是 af;每个结点的两个子结点均是该结点在 $\mathbb{Z}_m[x]$ 中的非平凡首一因子;叶结点均为 $\mathbb{Z}_m[x]$ 的不可约因子.*

注143. 实际我们用模素数的因子分解生成因子树时, 需要用扩展 Euclid 算法算出各个属于同一中间结点的两个子结点的 Bezout 系数, 以便进行下面的多因子 Hensel 提升算法. 一个典型的因子树如图10.1所示. 其中每个结点的括号中显示了其相应的 Bezout 系数.

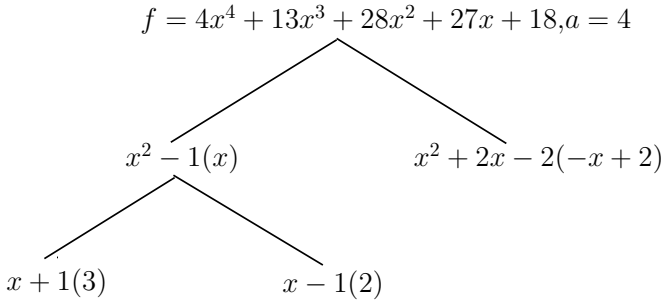

图 10.1: 模 5 因子树

当然由模 m 的因子树我们可以由单步 Hensel 算法得到模 m^2 乃至更高次幂的因子树, 只要我们由根结点依次在每个结点做 Hensel 提升即可. 下面给出该算法:

算法10.4 (多因子 Hensel 提升).

输入: 整数 m 和 n 次多项式 $f \in \mathbb{Z}[x]$, $a_0 \in \mathbb{Z}$ 使得 $a_0 \mathrm{lc}(f) \equiv 1 \pmod{m}$, 正整数 l, 一个 f 模 m 的因子树 τ, 共有 r 个叶子.

输出: 整数 a^* 使得 $a^* \mathrm{lc}(f) \equiv 1 \pmod{m^l}$ 和一个 f 模 m^l 的因子树 τ^*, 各个新结点 v^* 和原结点 v 满足 $v^* \equiv v \pmod{m}$.

1. $d = \lceil \log_2 l \rceil$, $\tau_0 = \tau$,
2. 对 j 从 1 循环到 d, 执行第 3 至 5 步,
3. 计算整数 a_j 使得 $a_j \equiv 2a_{j-1} - \mathrm{lc}(f) a_{j-1}^2 \pmod{m^{2^j}}$, $\tau_j = \tau_{j-1}$, 将 τ_j 的根结点换为 $a_j f$,
4. 从根结点遍历 τ_j 的结点, 对每个非叶结点 v, 执行第 5 步(由根向叶结点方向进行),
5. 调用算法 10.3 , 输入 $m^{2^{j-1}}$ 来提升 $v = g_v h_v$ 和 $s_v g_v + t_v h_v \equiv 1 \pmod{m^{2^{j-1}}}$, 提升到模 m^{2^j},
6. 输出 a_d, τ_d.

注144. 注意到 d 的选取能够使 $2^d \geq l$, 即我们已将因子树提升到足够高的次数.

注145. 为说明算法的有效性, 我们只需要说明每一步求得的 a_j 确实满足要求. 这是因为 $a_j \mathrm{lc}(f) \equiv 2a_{j-1}\mathrm{lc}(f) - \mathrm{lc}^2(f)a_{j-1}^2 \equiv 1 - (1 - a_{j-1}\mathrm{lc}(f))^2 \equiv 1 \pmod{m^{2^j}}$.

10.3 应用 Hensel 提升的 Zassenhaus 算法

有了前面的 Hensel 提升理论, 我们就可以利用它取代大素数模算法. Zassenhaus[189] 在 1969 年将 Hensel 提升算法引入到整系数多项式因子分解算法中, 下面我们就来介绍 Zassenhaus 算法.

算法10.5 (整系数多项式分解 3:素数幂和因子组合算法).

输入: 一个无平方因子本原 n 次多项式 $f \in \mathbb{Z}[x]$, $\mathrm{lc}(f) > 0$ 且 $\|f\|_\infty = A$,

输出: f 的不可约因子 $\{f_1, \ldots, f_k\} \subset \mathbb{Z}[x]$.

1. 若 $n = 1$ 则输出 f, 否则 $b = \mathrm{lc}(f)$, $B = (n+1)^{1/2}2^n Ab$, $C = (n+1)^{2n}A^{2n-1}$, $\gamma = \lceil 2\log_2 C \rceil$,
2. 任意选取素数 $p \leq 2\gamma \ln \gamma$, $\overline{f} = f \bmod p$, 直到 $p \nmid b$ 且 $\gcd(\overline{f}, \overline{f}') = 1 \in \mathbb{F}_p[x]$, 然后令 $l = \lceil \log_p(2B+1) \rceil$,
3. 调用有限域上因子分解算法计算 $h_1, \ldots, h_r \in \mathbb{Z}[x]$, 各因子均是首一不可约因子, 且无穷范数小于 $p/2$, 于是 $f \equiv bh_1 \cdots h_r \pmod{p}$,
4. $a = b^{-1} \bmod p$, 利用扩展 Euclid 算法建立 f 模 p 的因子树, 叶结点为 $h_1, \ldots, h_r$, 再调用算法 10.4 计算分解 $f \equiv bg_1 \cdots g_r \pmod{p^l}$, 其中 $g_i \in \mathbb{Z}[x]$ 为首一且无穷范数小于 $p^l/2$, $g_i \equiv h_i \pmod{p}$,
5. 调用因子组合算法并输出. (此处同算法 10.1 第 4 ∼ 10 步.) $T = \{1, \ldots, r\}$, $s = 1$, $G = \varnothing$, $f^* = f$,
6. 当 $2s \leq \#T$ 时循环执行下面 4 步, 否则转第 11 步,
7. 枚举 T 的所有 s 元子集 S, 并做下两步 8, 9 循环:
8. 计算 $g^*, h^* \in \mathbb{Z}[x]$ 使得其无穷范数比 $p^l/2$ 要小并且 $g^* \equiv b\prod_{i\in S} g_i \pmod{p^l}$, $h^* \equiv b\prod_{i\in T\setminus S} g_i \pmod{p^l}$,
9. 若 $\|g^*\|_1\|h^*\|_1 \leq B$ 则 $T = T\setminus S$, $G = G\bigcup\{\mathrm{pp}(g^*)\}$, $f^* = \mathrm{pp}(h^*)$, $b = \mathrm{lc}(f^*)$, 跳出 7, 8, 9 循环并转第 6 步,

10. $s = s + 1$,

11. 输出 $G \bigcup \{f^*\}$.

注146. 步骤 2 中 γ 的引入见 [174]18.4 节.

注147. 关于素数的选取, 我们也可以由序列 $3, 5, 7, \ldots$ 依次选取, 或者预先准备好小素数表. 为了降低因子组合的复杂性, 选取 p 并进行分解

$$f \equiv bh_1 \cdots h_r \pmod{p}$$

后, 我们可以多选几个适合的 p 分解, 取 r 最小的那个 p 进行后面的 Hensel 提升.

注148. 我们还可以在进行 Hensel 提升的过程中一边检查各个因子是不是 f 在 $\mathbb{Z}[x]$ 中的因子, 如果是则将其从树中移除, 仅提升剩余的因子. 这样, 我们有可能在提升到 l 次幂之前就已经分解完全了.

算法有效性. 记 f 的因子 $u \in \mathbb{Z}[x]$, 其在 $\mathbb{F}_p[x]$ 中不可约因子个数 $\mu(u)$. 现在我们要归纳证明在每次到第 6 步时, 有下面命题成立:

1. $f^* \equiv b\prod_{i\in T} g_i \pmod{p^l}$, $\quad b = \mathrm{lc}(f^*)$, $\quad f = f^* \prod_{g\in G} g$,

2. G 中多项式均不可约,

3. f^* 本原且它的任何一个不可约因子 $u \in \mathrm{Z}[x]$ 有 $\mu(u) \geq s$.

主要证法和算法 10.1 的证明一致, 我们只需证明当 f^* 有一个不可约因子 g 满足 $\mu(g) = s$ 时, 当循环到 s 时必然能将此因子选出, 这一点可以构造来证明, 即取指标集 S 为 g 在 $\mathbb{F}_p[x]$ 中不可约因子的编号. 当然这里与前面的证法有所不同, 因为 $\mathbb{Z}_{p^l}[x]$ 并不是 UFD, 要证明唯一性还需要用 Hensel 提升的唯一性定理. 首先可设 $\mathrm{lc}(h)g \equiv b\prod_{i\in S} g_i \pmod{p}$, $\mathrm{lc}(g)h \equiv b\prod_{i\in T\setminus S} g_i \pmod{p}$, 再设 $g^* \equiv b\prod_{i\in S} g_i \pmod{p^l}$, $h^* \equiv b\prod_{i\in T\setminus S} g_i \pmod{p^l}$. 由 $f^* = gh$ 可知

$$bf^* = \mathrm{lc}(h)g\,\mathrm{lc}(g)h,$$

另外

$$bf^* \equiv g^*h^* \pmod{p^l},$$

以上两式均是 $bf^* \bmod p$ 的提升, 于是 $\mathrm{lc}(h)g \equiv g^* \pmod{p^l}$ 且 $\mathrm{lc}(g)h \equiv h^* \pmod{p^l}$, 再由所定的 Mignotte 界和 l 的选择可知 9 中的条件必然成立. □

例10.4. 仍然考虑例 10.1 中 $f=4x^4+13x^3+28x^2+27x+18\in\mathbb{Z}[x]$ 的分解.

解: 首先, $n=4$, $A=28$, $b=4$, $B=1792\sqrt{5}=4007.03$, $\gamma=104$, 而 $2,3\mid \mathrm{disc}(f)$, 故取 $p=5$, 此时 $l=\lceil\log_5(2B+1)\rceil=6$, 要提升 3 次. 首先利用模 5 因子分解和扩展 Euclid 算法得到如下模 5 因子树(图10.1):

$$4f=\begin{cases} s=x, g=x^2-1 \begin{cases} s=3, g=x+1 \\ t=2, h=x-1 \end{cases} \\ t=-x+2, h=x^2+2x-2 \end{cases}$$

利用多因子提升算法提升为模 25 因子树:

$$19f=\begin{cases} s=6x, g=x^2-5x-11 \begin{cases} s=-2, g=x-9 \\ t=2, h=x+4 \end{cases} \\ t=-6x-8, h=x^2+2x+3 \end{cases}$$

以此类推有模 $5^4=625$ 因子树:

$$469f=\begin{cases} s=-69x, g=x^2-155x-311 \begin{cases} s=-177, g=x-134 \\ t=177, h=x-21 \end{cases} \\ t=69x-208, h=x^2+2x+3 \end{cases}$$

模 $5^8=390625$ 因子树:

$$292969f=\begin{cases} s=86806x, g=x^2-97655x-195311 \begin{cases} s=-108927, g=x+82991 \\ t=108927, h=x-180646 \end{cases} \\ t=-86806x-130208, h=x^2+2x+3 \end{cases}$$

于是我们有 $f\equiv 4(x+82991)(x-180646)(x^2+2x+3)\pmod{5^8}$. 这里我们再对其进行还原时, 显然有 $s=1$ 时可得到不可约因子 (x^2+2x+3), 此时 $f^*=h^*\equiv 4(x+82991)(x-180646)\equiv 4x^2+5x+6\pmod{5^8}$. 于是再一次得到分解 $f=(x^2+2x+3)(4x^2+5x+6)$. ◇

10.4　格中短向量理论

10.4.1　问题的引入

通过大素数模方法或 Hensel 提升方法, 我们都可以得到多项式 $f\in\mathbb{Z}[x]$ 在模某个整数 m 后的分解, 即 $f\equiv bg_1\cdots g_r\pmod m$. 这时候我们用因子组合的算法

将它们拼起来还原. 很显然, 因子组合算法的时间复杂度是指数级的, 在某些情况下这种方法的效率很低, 如 Swinnerton-Dyer 多项式(见 [174]15.3 节), 这个例子是最坏的情况, 即多项式不可约但是在有限域上却分解为一次不可约因子的乘积, 组合时要尝试所有 2^n 种情况才能得到结果. 下面几小节将要介绍的格中短向量方法是一种多项式时间算法, 它从理论上改进了原先因子组合算法的指数时间复杂度, 尽管我们实际仍然使用因子组合算法([56]3.5.5 节).

按照定义 8.13 , 以下我们取默认的范数 $\|f\|$ 为 2-范数, 即 $\|f\|_2$. 我们将多项式的系数看作列向量, 则多项式范数等同于该向量的范数. 首先我们有:

定理10.4 (Hadamard 不等式)**.** *设 n 阶方阵 A 用 n 个列向量表示为*

$$A = (a_1, a_2, \ldots, a_n),$$

则 $\deg A \leq \|a_1\|\|a_2\|\cdots\|a_n\|$.

证明参见高等代数学教材, 如可参考 [12]278 页 48 题.

引理10.3. *设 f, $g \in \mathbb{Z}[x]$ 分别是 n, k 次多项式, $u \in \mathbb{Z}[x]$ 是一非平凡首一多项式, m 为一正整数, 若在 $\mathbb{Z}_m[x]$ 中有 $u \mid f \pmod m$, $u \mid g \pmod m$, $\|f\|^k\|g\|^n < m$. 则 f 和 g 有非平凡公因子.*

证明. 假设 f, g 没有非平凡的公因子, 则在 $\mathbb{Q}[x]$ 中有 $\gcd(f,g) = 1$, 由结式理论推论 8.4 可知存在 s, $t \in \mathbb{Z}[x]$ 使得 $sf + tg \equiv \operatorname{res}(f,g) \pmod m$, 由于 u 在模 m 下能整除 f 和 g, 则 $u \mid \operatorname{res}(f,g) \bmod m$. 但 u 是非平凡首一多项式, 于是必有 $\operatorname{res}(f,g) \equiv 0 \pmod m$, 再由 Hadamard 不等式有 $|\operatorname{res}(f,g)| < \|f\|^k\|g\|^n < m$, 此即说明 $\operatorname{res}(f,g) = 0$, 与 $\gcd(f,g) = 1 \in \mathbb{Q}[x]$ 矛盾. 于是二者在 $\mathbb{Z}[x]$ 中有非平凡公因子. □

由前面我们已得出的分解结果和上面的引理, 我们产生如下的分解的想法. 首先我们已经有待分解的 n 次多项式 $f \in \mathbb{Z}[x]$ 和 f 在模 m 下的一个因子 $u \in \mathbb{Z}[x]$, 此时我们需要找到一个较"短"的 k 次多项式 $g \in \mathbb{Z}[x]$ 使得 $\|g\|^n < m\|f\|^{-k}$, 且 $u \mid g \bmod m$, 于是可通过 $\gcd(f,g)$ 得到 f 的一个非平凡因子. 为了叙述方便, 以后记号 f 既可以表示 n 次多项式, 也可以表示 $n+1$ 维系数列向量. 引入下面的定义:

定义10.3. 设有正整数 n 和向量 $f_1, \ldots, f_n \in \mathbb{R}^n$, 则

$$L = \sum_{1\leq i\leq n} \mathbb{Z}f_i = \{ \sum_{1\leq i\leq n} r_i f_i \mid r_1, \ldots, r_n \in \mathbb{Z}\}$$

称为由 $f_1, \ldots, f_n$ 生成的格(Lattice). $f_1, \ldots, f_n$ 称为 L 的基. 而 L 的范数是定义为

$$|L| = |\det(f_1, \ldots, f_n)| \in \mathbb{R}.$$

后面将看到范数的定义是与基的选择无关的.

现在我们考虑寻找一个次数小于 j 的多项式 $g \in \mathbb{Z}^j$, 设 $L \subset \mathbb{Z}^j$ 是由 u 和 m 生成的, 即

$$L = \{g = qu + rm \mid q, r \in \mathbb{Z}[x], \deg q < j - d, \deg r < d\},$$

其中 $d = \deg u$. 这时我们有下面的定理:

定理10.5. *对于任何一个次数小于 j 的多项式 g, $u \mid g \bmod m \Leftrightarrow g \in L$.*

证明. 对于 $\Rightarrow$ 显然, 对于 $\Leftarrow$, 我们有 $g = q^*u + r^*m$, 再由 r^* 对首一多项式 u 的除法可得 $r^* = q^{**}u + r^{**}$, 其中 $\deg r^{**} < \deg u$. 令 $q = q^* + mq^{**}$, $r = r^{**}$, 则有 $g = qu + rm$, 且 $\deg q < j - d$, $\deg r < d$, 于是 $g \in L$. □

现在我们的问题化为在 L 中寻找一种约化的基, 以使得基向量长度较短, 满足要求.

10.4.2 约化基算法

下面要介绍的约化基算法即是所谓的 3-L(Lenstra, Lenstra and Lovász)算法.

引理10.4. *令 $N \subset M \subset \mathbb{R}^n$ 为两个格, 分别由 $g_1, \cdots g_n$ 和 $f_1, \ldots, f_n$ 生成, 则 $|M|$ 整除 $|N|$.*

证明. 由 $N \subset M$ 知 N 的生成元 g_i 均是 M 的生成元 f_i 的线性组合, 即存在整系数矩阵 A 使得 $(g_1, \ldots, g_n) = A(f_1, \ldots, f_n)$, 于是 $|N| = \det A|M|$, $\deg A \in \mathbb{Z}$. □

取 $N = M$ 可知格的范数与生成元无关. 由 Hadamard 不等式我们知道 $|M| \leq \|f_1\| \cdots \|f_n\|$.

因为范数与基的选择无关, 因而我们想到, 如果选取的基越"正交", 那么某种程度上这组基向量的长度越短. 因而, 向量基的正交化可以启示我们得到一种求约化基(即所谓的短向量)的方法.

现在我们已知由 $f_1, \ldots, f_n$ 生成的格, 若取内积为 $\langle f, g\rangle = f^T g$, 由高等代数学(例如参见 [12]281 页)的内容我们知道可以对它们进行 Gram-Schmidt 正交化, 正交化的过程可以归纳地进行, 即令 $f_1^* = f_1$, 对于 $i \geq 2$, 有

$$f_i^* = f_i - \sum_{1 \leq j < i} f_j^* \mu_{ji}, 其中 \mu_{ji} = \frac{\langle f_i, f_j^*\rangle}{\langle f_j^*, f_j^*\rangle}.$$

于是存在一个上三角阵 $M=(\mu_{ij})$, 其对角元为 1, 使得

$$(f_1,\dots,f_n)=(f_1^*,\dots,f_n^*)M,$$

其中 f_i^* 张成同样的空间, 且两两互相正交. 由正交化的几何意义我们可以很明显地看到, 每个向量的长度都不大于原向量的长度, 即基向量的长度缩短了. 但是光作 GSO(Gram-Schmidt orthogonalization)是不行的, 因为正交化后所得的向量的组合系数可以为有理数, 并不一定是原先格中的向量. 但是首先, 我们有下面的估计:

引理10.5. *设 L 为由 $(f_1,\dots,f_n)$ 生成的格, $(f_1^*,\dots,f_n^*)$ 是对其进行 GSO 所得的基, 则对任意非零向量 $f\in L$, 我们有*

$$\|f\|\geq\min\{\|f_1^*\|,\dots,\|f_n^*\|\}.$$

证明. 由 GSO 过程知道有对角元为 1 的上三角阵 $M=(\mu_{ij})$ 满足

$$(f_1,\dots,f_n)=(f_1^*,\dots,f_n^*)M,$$

则 $f_i=\sum_{1\leq j\leq n}f_j^*\mu_{ji}=\sum_{1\leq j\leq i}f_j^*\mu_{ji}$. 对于非零向量 $f\in L$, 存在 $k\leq n$ 使得

$$f=\sum_{1\leq i\leq k}\lambda_i f_i,\quad \lambda_k\neq 0,\lambda_1\cdots\lambda_k\in\mathbb{Z}.$$

于是 $f=\sum_{1\leq i\leq k}\lambda_i f_i=\sum_{1\leq i\leq k}\lambda_i\sum_{1\leq j\leq i}f_j^*\mu_{ji}=\lambda_k f_k^*+\sum_{1\leq i<k}\gamma_i f_i^*$, 其中 $\gamma_i\in\mathbb{R}$, 将此式代入 $\|f\|$ 可得:

$$\begin{aligned}\|f\|^2&=(\lambda_k f_k^*+\sum_{1\leq i<k}\gamma_i f_i^*)(\lambda_k f_k^*+\sum_{1\leq i<k}\gamma_i f_i^*)\\&=\lambda_k^2\|f_k^*\|^2+\sum_{1\leq i<k}\gamma_i^2\|f_i^*\|^2\geq\lambda_k^2\|f_k^*\|^2\\&\geq\|f_k^*\|^2\geq\min\{\|f_1^*\|^2,\dots,\|f_n^*\|^2\}.\end{aligned}$$

证毕. □

既然我们用 GSO 得到的不一定是格中的向量, 那么我们可以放宽条件, 下面给出一种约化基的定义方式:

定义10.4. 设 $f_1,\dots,f_n\in\mathbb{R}^n$ 线性无关且 $(f_1^*,\dots,f_n^*)$ 是其对应的 GS 正交基. 则称 $(f_1,\dots,f_n)$ 是约化基, 如果对于 $1\leq i<n$ 有 $\|f_i^*\|^2\leq 2\|f_{i+1}^*\|^2$ 且对于 μ_{ji} 的非零元均有 $|\mu_{ji}|<1/2$.

下面给出的定理对于后面利用约化基算法进行因子分解是有用的.

定理10.6. 设 $(f_1,\dots,f_n)$ 是其所生成的格 $\in \mathbb{R}^n$ 的约化基且 $f \in L\setminus\{0\}$, 则 $\|f_1\| \le 2^{(n-1)/2}\|f\|$.

证明. 由于

$$\|f_1\|^2 = \|f_1^*\|^2 \le 2\|f_2^*\|^2 \le 2^2\|f_3^*\|^2 \le \cdots \le 2^{n-1}\|f_n^*\|^2,$$

可设 $\|f_m^*\| = \min\{\|f_1^*\|,\dots,\|f_n^*\|\}$, 则由引理 10.5 可得

$$2^{n-1}\|f\|^2 \ge 2^{n-1}\|f_m^*\|^2 \ge 2^{m-1}\|f_m^*\|^2 \ge \|f_1^*\|^2 = \|f_1\|^2,$$

于是命题得证. □

下面给出生成约化基的算法:

算法10.6 (约化基算法).

输入: 线性无关的向量 $f_1,\dots,f_n \in \mathbb{Z}^n$,

输出: 由输入向量所生成的格的约化基 $(g_1,\dots,g_n)$.

1. 将 $g_1,\dots,g_n$ 初始化为 $f_1,\dots,f_n$, 并进行相应的 GSO 过程, 得到约化基 $G^* = (g_1^*,\dots,g_n^*)$ 和变换矩阵 $M \in \mathbb{Q}^{n\times n}$ 使得 $G = G^*M$, $i = 2$,
2. 当 $i \le n$ 时, 执行下面 3 ∼ 5 步,
3. 让 j 从 $i-1$ 循环到 1, 执行下面 4 步,
4. $g_i = g_i - g_j\lceil\mu_{ji}\rfloor$(最接近 μ_{ji} 的整数), 重新计算 GSO 得到 G^*, M,
5. 若 $i > 1$ 且 $\|g_{i-1}^*\|^2 > 2\|g_i^*\|^2$ 则交换 g_{i-1} 和 g_i, 并重新计算 GSO 得到 G^*, M, $i = i-1$, 否则 $i = i+1$,
6. 输出 $(g_1,\dots,g_n)$.

注149. 也可以由 GSO 过程求得变换矩阵 M 使得 $(f_1^*,\dots,f_n^*) = (f_1,\dots,f_n)M$, 再将 M 中元素均取整后代替算法 10.6 第 4 步中的计算. 下面举的几个例子用此算法, 当然算法 10.6 中的矩阵 M 更好计算一些.

例10.5. 记 $a = 219914302784468853031851163930189578018051741$, $b = 5^{64}$, 设 L 由 $(1,a)$ 和 $(0,b)$ 生成, 用上面的算法求其约化基.

解: 通过计算可得如下约化基:

$$g_1 = (-6999570441183108942993, -13384313532767302775813),$$

$$g_2 = (-3253854280751988427427 3, 15228795581564048106332),$$

其中

$$g_1 = -6999570441183108942993(1, a) + 28395177438811868116 24(0, b).$$

我们取 $g^* = g_1$ 作为短矢量, 有

$$g^* = -6999570441183108942993x - 13384313532767302775813.$$

如果我们再考虑由 $(1, a, 0), (0, 1, a), (0, 0, b)$ 生成的格, 则按照算法可得如下约化基:

$$g_1 = (4, 5, 6),$$

$$g_2 = (5989804332598408382848, 48768497456490153556 7, -4399607033869690201541),$$

$$g_3 = (-3921135160431868252895, 6733001971931690223946, -2996744869655163018019),$$

其中

$$g_1 = 4(1, a, 0) - 879657211137875412127404655720758312072206959(0, 1, a) + 3568507925661854502695244169691361191689403 13(0, 0, b),$$

此时得到短矢量 $g^* = 4x^2 + 5x + 6$. ◇

10.4.3 约化基算法的一些细节说明

约化基算法 10.6 大致说明了生成约化基的思路, 实际计算约化基时我们需要对它的细节描述推敲一下. 例如仔细分析一下我们会发现该算法第 4 步中的 GSO 更新是不必要的, 可以将这一步省去. 而且在每一步交换两个基向量时对 GSO 的更新也是有很多冗余计算的.

事实上, 关于格的约化基的定义方式在各文献中不一致, 上小节所提的定义方式是 [174] 给出的. A. K. Lenstra, H. W. Lenstra 和 L. Lovász[116] 最初给出如下定义:

定义10.5. 设 $f_1, \ldots, f_n \in \mathbb{R}^n$ 线性无关且 $(f_1^*, \ldots, f_n^*)$ 是其对应的 GS 正交基, 并且记 $\mu_{ij} = \langle f_i, f_j^* \rangle / \langle f_j^*, f_j^* \rangle$ 则称 $(f_1, \ldots, f_n)$ 是约化基, 如果满足以下条件:

1. $|\mu_{ij}| \le 1/2, \forall 1 \le j < i \le n$,
2. $|f_i^* + \mu_{i,i-1} f_{i-1}^*|^2 \ge \frac{3}{4}|f_{i-1}^*|^2, \forall 1 < i \le n$.

注150. 这里关于矩阵元 μ_{ij} 的定义与上小节有略微不同, 这是由于上小节将 f_i 视为列向量, 而本节视为行向量, 只是叙述语言的不同, 并无本质差别.

从定义中我们明显看出, 本小节定义的约化基一定是上一小节定义的约化基, 其条件更强, 因此对后面的算法是没有影响的. 下面我们给出新的约化基算法, 它在细节处理上比前节的算法要节省很多计算.

算法10.7 (约化基算法).

输入输出同算法 10.6 .

1. 初始化工作. 首先将各 g_i 初化为 f_i, 并且 i 顺次由 1 循环到 n, 执行第 2 至 4 步,
2. $g_i^* = g_i$,
3. 将 j 顺次由 1 循环到 $i-1$, 计算 $\mu_{ij} = \langle g_i, g_j^* \rangle / G_j$, $g_i^* = g_i^* - \mu_{ij} g_j^*$,
4. $G_i = \langle g_i^*, g_i^* \rangle$,
5. $k = 2$, 并循环做下面 6–10 步,
6. 将 l 顺次由 $k-1$ 递减到 1, 做如下 7–8 步,
7. 若 $|\mu_{kl}| > 1/2$, 则
$$r = \lfloor \mu_{kl} \rceil, g_k = g_k - r g_l,$$
对 $j = 1, 2, \ldots, l-1$ 顺次计算 $\mu_{kj} = \mu_{kj} - r\mu_{lj}$, 并且 $\mu_{kl} = \mu_{kl} - r$,
8. 若 $l = k-1$ 则检验 $G_k < (\frac{3}{4} - \mu_{k,k-1}^2) G_{k-1}$ 是否成立, 若成立则交换 g_k 与 g_{k-1} 并更新此时的各 GSO 结果, 并直接转第 6 步,
9. 若 $k = n$ 则终止算法并输出 $(g_1, \ldots, g_n)$,
10. $k = k + 1$.

算法的终止性证明可参见 [116].

本算法仅在初始化时需要计算诸 g_i^*, 在后面的算法中实际上已不需要它们, 只要保存 μ_{ij} 矩阵以及各 g_i^* 的模 G_i 即可. 在本节的开始, 我们也提到了在交换两个

基矢时对整个 GSO 的更新实际上包含了很多不必要的计算, 因此本算法第 8 步所提的到的更新 GSO 步骤是很重要的, 我们在下面给出:

算法10.8 (约化基算法第八步的更新).

各符号同算法 10.7 .

1. $\mu = \mu_{k,k-1}$, $G = G_k + \mu^2 G_{k-1}$, $\mu_{k,k-1} = \mu G_{k-1}/G$, $G_k = G_{k-1}G_k/G$, $G_{k-1} = G$,
2. 交换两个基矢 $(g_{k-1}, g_k) = (g_k, g_{k-1})$,
3. 对 $j = 1, 2, \ldots, k-2$ 顺次交换 $(\mu_{k-1,j}, \mu_{kj}) = (\mu_{kj}, \mu_{k-1,j})$,
4. 对 $i = k+1, k+2, \ldots, n$ 顺次计算

$$\begin{pmatrix} \mu_{i,k-1} \\ \mu_{ik} \end{pmatrix} = \begin{pmatrix} 1 & \mu_{k,k-1} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -\mu \end{pmatrix} \begin{pmatrix} \mu_{i,k-1} \\ \mu_{ik} \end{pmatrix}$$

5. 若 $k > 2$ 则 $k = k - 1$.

更新算法的正确性可通过一些计算证明, 限于篇幅, 我们也不在这里验证了, 感兴趣的读者可以参考 [116]520 页.

10.5 应用格中短向量的分解算法

下面以 Hensel 提升和格中短向量方法为例说明约化基算法如何用于因子分解. 这里我们先证明一个引理, 用于后面算法正确性的证明.

引理10.6. *$p \in \mathbb{Z}$ 是素数, l 是正整数, f, g, $u \in \mathbb{Z}[x]$ 是非零多项式且 $p \nmid \mathrm{lc}(f)$, $f \bmod p$ 无平方, $g \mid f$, u 是首一非平凡多项式, 且 $u \mid f \bmod p^l$, $u \mid g \bmod p$. 则 $u \mid g \bmod p^l$.*

证明. 令 $h, v, w \in \mathbb{Z}[x]$ 使得 $f = gh \equiv uw \pmod{p^l}$ 且 $g \equiv uv \pmod{p}$. $f \bmod p$ 无平方, 则 $g \bmod p$ 也无平方因子, $\gcd(u \bmod p, v \bmod p) = 1 \in \mathbb{F}_p[x]$. 由 Hensel 引理知有 u^*, $v^* \in \mathbb{Z}[x]$ 使得 $u^* \equiv u \pmod{p}$, $v^* \equiv v \pmod{p}$ 且 $g \equiv u^*v^* \pmod{p^l}$. 于是 $uvh \equiv gh \equiv uw \pmod{p}$ 推出 $vh \equiv w \pmod{p}$. $v^*h \equiv vh \equiv w \pmod{p}$ 且 $u^*(v^*h) \equiv gh = f \equiv uw \pmod{p^l}$. 再由 u, v 在模 p 下互素, 我们由提升的唯一性知 $u^* \equiv u \pmod{p^l}$, $g \equiv uv^* \pmod{p^l}$. □

下面给出算法 [174]477 页.

算法10.9 (整系数因子分解算法 4:Hensel 提升和格中短向量算法).

输入: 一个无平方因子本原 $n(\geq 1)$ 次多项式 $f \in \mathbb{Z}[x]$, $\mathrm{lc}(f) > 0$ 且 $\|f\|_\infty = A$,

输出: f 的不可约因子 $\{f_1, \ldots, f_k\} \subset \mathbb{Z}[x]$.

1. 若 $n = 1$ 则输出 $\{f\}$, 否则 $b = \mathrm{lc}(f)$, $B = (n+1)^{1/2} 2^n A$, $C = (n+1)^{2n} A^{2n-1}$, $\gamma = \lceil 2 \log_2 C \rceil$,
2. 任选素数 $p \leq 2\gamma \ln \gamma$, $\overline{f} = f \bmod p$, 直至 $p \nmid b$ 且 $\gcd(\overline{f}, \overline{f}') = 1 \in \mathbb{F}_p[x]$, 再令 $l = \lceil \log_p(2^{n^2} B^{2n}) \rceil$,
3. 在 $\mathbb{F}_p[x]$ 上得到分解 $f \equiv b h_1 \cdots h_r \pmod p$, 各因子无穷范数小于 $p/2$ 且首一,
4. 利用 Hensel 提升得到分解 $f \equiv b g_1 \cdots g_r \pmod{p^l}$, 各因子不可约且无穷范数小于 $p^l/2$, 首一,
5. $T = \{1, \ldots, r\}$, $G = \varnothing$, $f^* = f$,
6. 当 $T \neq \varnothing$ 时, 做下面 7 ~ 10 步,
7. 在 $\{g_t \mid t \in T\}$ 中选择次数最大的因子 u, $d = \deg u$, $n^* = \deg f^*$, 对 $d < j \leq n^*$ 的 j 循环做下面 8, 9 步,
8. 调用算法 10.6 计算一个短矢量 $g^* \in L$, 其中 L 为

$$\{u x^i \mid 0 \leq i < j - d\} \cup \{p^l x^i \mid 0 \leq i < d\},$$

9. 用试除法得到 $S \subset T$, 其中 $S = \{i \in T | h_i | g^* \bmod p\}$, 计算 $h^* \in \mathbb{Z}[x]$ 使得其无穷范数小于 $p^l/2$ 且满足 $h^* \equiv b \prod_{i \in T \setminus S} g_i \pmod{p^l}$, 若 $\|\mathrm{pp}(g^*)\|_1 \|\mathrm{pp}(h^*)\|_1 \leq B$ 则 $T = T \setminus S$, $G = G \bigcup \{\mathrm{pp}(g^*)\}$, $f^* = \mathrm{pp}(h^*)$, $b = \mathrm{lc}(f^*)$, 转第 6 步,
10. $T = \varnothing$, $G = G \bigcup \{f^*\}$,
11. 输出 G.

注151. 算法复杂度为 [174]

$$O\left(n^6(n+\log A)M(n^2(n+\log A))(\log n+\log\log A)\right).$$

算法有效性: 只需证明每次执行第 6 步时, 我们有

$$f^*\equiv b\prod_{i\in T}g_i\pmod{p^l},\quad b=\mathrm{lc}(f^*),\quad f=\pm f^*\prod_{g\in G}g,$$

且 G 中多项式均不可约. 初始时显然满足, 假设每次到第 7 步时均满足, 令 $g\in\mathbb{Z}[x]$ 为 f^* 的一个不可约因子且 $u\mid g \bmod p$, 则由本节开始的引理知 $u\mid g \bmod p^l$. 相反地, 若 $v\in\mathbb{Z}[x]$ 是 f 的因子且在模 p 下能被 u 整除, 则 $g\mid v\in\mathbb{Z}[x]$. 我们可以证明第 9 步中的条件成立当且仅当 $\mathrm{pp}(g^*)\mathrm{pp}(h^*)=\pm f^*$, 又 $\deg g^*<j$, 我们知道只要 $j\le\deg g$ 则条件不满足. 同样地, 第 10 步能使得上述条件在算法循环结束时也是成立的, 如果 $\#T=1$ 且 $g=f^*$ 是不可约的.

假设 $\deg g<n^*$, 令 $j=1+\deg g$. 则 $g\in L$. 于是由定理 10.6 有 $\|g^*\|\le 2^{(j-1)/2}\|g\|<2^nB$. 再由 l 的选择, 我们有

$$\|g^*\|^{j-1}\|g\|^{\deg g^*}<(2^nB)^nB^n\le p^l,$$

于是由引理 10.3 知 $\gcd(g,g^*)$ 非平凡, 由 g 不可约以及 $\deg g^*\le j-1=\deg g$ 得 $g=\pm\mathrm{pp}(g^*)$.

令 $h=f^*/g$, 则由 Hensel 提升唯一性知在第 9 步有 $\mathrm{lc}(g)h\equiv h^*\pmod{p^l}$. 再由 $p^l/2$ 大于 $\|bh\|_\infty$ 的 Mignotte 界 bB, 于是 $\mathrm{lc}(g)h=h^*, h=\mathrm{pp}(h^*)$, 且 $f^*=\pm\mathrm{pp}(g^*)\mathrm{pp}(h^*)$, 算法会到到 f 的不可约因子 g. □

例10.6. 利用算法 10.9 分解例 10.1 和例 10.4 中的例子 $f=4x^4+13x^3+28x^2+27x+18\in\mathbb{Z}[x]$.

解: 首先, 此时 $l=\log_5(2^{n^2}B^{2n})=48.1266$, 故需提升 6 次, 我们在例 10.4 的结果上再提升三次, 可以得到如下结果:

$$f\equiv 4(x^2+2x+3)(x+219914302784468853031851163930189578018051741)$$
$$(x+18666151189759530972094363639603856376661622 9)\pmod{5^{64}},$$

另外由前面例 10.4 结果我们知道

$$f\equiv 4(x^2+2x-2)(x+1)(x-1)\pmod 5.$$

首先 $u=x^2+2x+3$, $d=2$, 则 L 的生成元为

$$\{(1,2,3),(0,5^{64},0),(0,0,5^{64})\}.$$

很显然 $(1,2,3)$ 应该是一个短向量, 由试除法得到 $S=\{1\}$, $h^*=4\prod_{i=2,3}g_i \bmod 5^{64}=4x^2+5x+6$, 显然满足条件, 得到一个不可约因子 x^2+2x+3, 此时 $f^*=4x^2+5x+6$, $b=4$. 此时我们再取因子

$$u=x+219914302784468853031851163930189578018051741,$$

$d=1$, $n^*=2$, j 只能取 2, 则得到 L 的生成元 $\{u,(0,5^{64})\}$, 由例 10.5 得到

$$g^*=-69995704411831089429 93x-13384313532767302775813,$$

g^* 是本原的且是 $g^*\equiv 2x+2 \pmod 5$, 由试除法知 $S=\{2\}$, 但此时已有 $\|g^*\|_1>B$, 故此时无解, 循环结束, 第二个不可约因子为此时的 $f^*=4x^2+5x+6$.

其实我们第一步如果不取最大次数的 u, 可以对短向量方法有更深的体会. 若取 $u=g_2$, 当 $j=3$ 时则由例 10.5 后半部分的讨论可知此时得到短向量 $g^*=4x^2+5x+6$. ◇

11 多元多项式

在前面的章节中, 我们已经具体地讨论了各种一元多项式环上的 GCD 问题以及因子分解问题, 本章我们讨论多元多项式的相关问题, 即在 $\mathbb{Z}[x_1, \ldots, x_n]$ 中讨论.

对于多元多项式的最大公因子和因子分解问题, 和一元问题类似, 我们也主要采用各种同态象的方法, 以求将多元问题转化为一元问题求解.

11.1 多元多项式插值方法

回忆我们在 $\mathbb{Z}[x]$ 中处理问题时所用的方法. 为了有效避免系数膨胀问题, 我们无论求最大公因子还是做因子分解, 都采用了模算法, 将其化为有限域上多项式问题, 并且最后用中国剩余定理, Hensel 提升以及若干特殊的技术(如因子组合, 格中短矢量等)将其恢复. 对于多元多项式环, 不仅要在 $\mathbb{F}_p$ 中讨论问题, 还要将多元问题化为一元问题, 这里我们主要采用赋值同态的方法. 即同态映射

$$\Phi_{x_i-a}(f) = f \bmod (x_i - a) = f(x_1, \ldots, x_{i-1}, a, x_{i+1}, \ldots, x_n).$$

这样相当于给某些未定元取赋值点, 最后我们需要用中国剩余定理, 或者说是插值的方法还原多项式. 为了便于后文叙述, 我们引入记号 $\Phi_I(f)$ 表示 $f \bmod I$, 其中 I 为多项式环中的理想. 对于整数 m 引入记号 $\Phi_m(f)$ 表示 $f \bmod m$.

前面我们曾经介绍过多点的 Lagrange 插值快速算法. 由于我们后面的算法会利用逐点插值, 甚至所谓的"稀疏插值", 因此本节将介绍稠密插值和稀疏插值算法(参考 [190]), 它们对于多元多项式最大公因子的模算法是很有用的.

11.1.1 稠密插值

插值问题要解决的问题是已知若干个点 $u_1, u_2, \ldots, u_k$ 和 $v_1, v_2, \ldots, v_k$, 求多项式 f 使得 $\forall i = 1, 2, \ldots, k$ 有 $f(u_i) = v_i$. 下面直接给出 Newton 插值算法, 很容易验证算法的正确性.

算法11.1 (Newton 插值).

输入和输出同前描述.

1. 赋初值 $f(x) = v_1$, $q(x) = (x - u_1)$,
2. 对于 $i = 2, 3, \ldots, k$, 顺次计算

$$f(x) = f(x) + \frac{q(x)(v_i - f(u_i))}{q(u_i)}, \quad q(x) = (x - u_i)q(x),$$

3. 输出 $f(x)$.

多元多项式的稠密插值基于的思想十分简单, 例如我们有一个函数 $F(x_1, \ldots, x_n)$, 可以求得它在某些点上的函数值, 我们的任务是找到一个各个变元次数均不超过 d 的多元多项式 $P(x_1, \ldots, x_n)$, 使得它们在各整点上的值相同. 注意到我们在这里的问题的提法是很具有一般性的, 不仅可以将 GCD 问题化为这种形式, 甚至也可以将诸如多项式的乘法等问题化为这种形式. 稠密插值的基本思想就是递归地依次将各个变元插值回来, 假设我们有一初值点 $(x_{10}, x_{20}, \ldots, x_{n0})$, 我们首先可以固定 $x_{20}, \ldots, x_{n0}$, 而再取 d 个 x_1 的值, 由一元插值方法(Newton 法或 Lagrange 法)求得多项式 $P(x_1, x_{20}, \ldots, x_{n0})$, 依次再确定各变元 $x_2, \ldots, x_n$ 即可. 若我们设 d 次一元多项式插值问题的复杂度为 $M(d)$, 则可知本算法的复杂度为 $O(n(d+1)^{n-1}M(d))$(参见 [191]).

11.1.2 稀疏插值

问题引入

我们先用一个例子来介绍稀疏插值算法, 以便对它有一个更好的了解.

从上节稠密插值算法过程我们可以看出, 对 n 个变元次数不超过 d 的多项式的插值我们大约要对函数 F 求值 $(d+1)^n$ 次. 例如 [191] 中所用的多项式

$$P(x, y, z) = x^5z^2 + x^5z + xy^4 + xyz^5 + y^5z,$$

其需要计算函数值 $(5+1)^3=216$ 次! 事实上, 这么多插值次数对于这样稀疏的多项式显然是极其不划算的. 我们重复一下插值的过程以说明稀疏插值是如何减少插值次数的.

首先可以由赋值点 $(x_i,y_0,z_0),i=0,\ldots,5$ 通过稠密插值得到 $P(x,y_0,z_0)=ax^5+bx+c$. 下一步即要再选择 5 个 y 的赋值点 y_i, 计算 $P(x,y_i,z_0)$ 才能由此插值得到 $P(x,y,z_0)$. 如果是稠密插值法, 此时对于每个 y_i, 我们都需取 6 个 x_i 来插值, 但是如果 y_i 选的值恰当(所谓恰当的意义, 后文定义 11.2 和定理 11.1 将会给出说明), 我们完全可以假设对于 $y_i(1\leq i\leq 5)$, 也有

$$P(x,y_i,z_0)=a_ix^5+b_ix+c,$$

于是, 对每个 y_i 只需取 3 个 x 的赋值点来插值, 通过解方程的方法计算系数 a_i,b_i,c_i.

这样, 我们总共计算函数值的次数为 $6+3\times 5=21$, 即可得到 $P(x,y,z_0)$, 注意从这一步开始, 我们已经节省了插值次数, 从原来的需要求值 $6\times 6=36$ 次降到了 21 次. 并且可设求得的 $P(x,y,z_0)$ 具有形式:

$$P(x,y,z_0)=ax^5+bxy^4+cxy+dy^5.$$

接下来的步骤就比较顺理成章了, 我们同样要再计算 $P(x,y,z_i),1\leq i\leq 5$, 此时可假设各多项式具有形式

$$P(x,y,z_i)=a_ix^5+b_ixy^4+c_ixy+d_iy^5,$$

则需再求值 $4\times 5=20$ 次, 总共求值次数 41 次, 比起 216 次已大大减少了.

稀疏插值算法

为了叙述方便, 给出如下记号. 设多项式 $P(X)=P(X_1,X_2,\ldots,X_n)\in F[X]$, F 为某个域, 每个变元 X_i 的次数都不超过 d, 并且 P 的的单项式项数为 T(对于稀疏多项式有 $T\ll(d+1)^n$). 设 $v=(v_1,\ldots,v_n)$ 为一 n 元有序对, 定义单项式记号

$$X^v=X_1^{v_1}X_2^{v_2}\cdots X_n^{v_n},$$

则多项式 P 可记为

$$P=c_1X^{e_1}+c_2X^{e_2}+\cdots+c_TX^{e_T},$$

其中 $e_1,\ldots,e_T$ 均为 n 元序对.

从某种意义上说, 集合 $\{e_1,e_2,\ldots,e_T\}$ 反映了多项式 P 的"结构", 因而我们定义下面的:

定义11.1 (模板). 记多项式指数的集合为

$$\mathrm{skel}P = \{e_1, e_2, \ldots, e_T\},$$

简称其模板(也可译作骨架, 见 [191]skeleton 定义). 并记其在前 k 维上的投影为

$$\mathrm{skel}_k P = \{(e_1, \ldots, e_k) | \exists e = (e_1, \ldots, e_T) \in \mathrm{skel}P\}.$$

定义11.2 (精确求值点). 设 $(x_1, \ldots, x_n) \in F^n$, 若 $\forall k(1 < k < n)$, 有

$$\mathrm{skel}P(X_1, \ldots, X_k, x_{k+1}, \ldots, x_n) = \mathrm{skel}_k P(X),$$

则称其为精确求值点(Precise Evaluation Point).

注意到一般情况下只有

$$\mathrm{skel}P(X_1, \ldots, X_k, x_{k+1}, \ldots, x_n) \subset \mathrm{skel}_k P(X),$$

欲使两者相等, 则必须 $x_{k+1}, \ldots, x_n$ 的取值满足以 $X_1, \ldots, X_k$ 为主变元的多项式 P 的各项系数($\in F[X_{k+1}, \ldots, X_n]$)均在其上不为零. 据此, 我们可以给出精确求值点的概率估计.

定理11.1 (精确求值点概率估计). 设 $P(X)$ 是一整环上的 n 元多项式, 其每个变元次数不超过 d, 总共有 T 项, 设 S 为一有限赋值点集合, $\#S = s$, 取求值点 $x = (x_1, \ldots, x_n) \in S^n$, 则其不是精确求值点的概率不超过

$$\frac{n(n-1)dT}{2s}.$$

证明. 根据引理 8.6 我们知道对于 P 以 $X_1, \ldots, X_k$ 为主变元的某个系数多项式来说, $x_{k+1}, \ldots, x_k$ 为其零点的概率不超过 $(n-k)d/s$. 因为系数最多有 T 项, 则概率不超过 $(n-k)dT/s$. 再对 $k = 1, 2, \ldots, n-1$ 求和即有概率不超过

$$\frac{(n-1)dT}{s} + \frac{(n-2)dT}{s} + \cdots + \frac{dT}{s} = \frac{n(n-1)dT}{2s}.$$

证毕. □

取足够大的 s 可以减小此概率, 这正是我们每一步稀疏插值时假设目标多项式具有同样的模板的概率依据.

下面, 我们把整个插值还原的过程描述如下:

算法11.2 (稀疏插值算法)**.**

输入:一个可以求值的函数 $f(X_1,\ldots,X_n)$, 精确求值点 $(x_{10},\ldots,x_{n0})$, 各变元次数均不超过的 d,

输出:多项式 $P(X_1,\ldots,X_n)\in F[X]$, 使得 P 和 f 在求值点上的值相同.

1. 随机任取 d 个值 $x_{11},x_{12},\ldots,x_{1d}$, 利用求得的值 $f(x_{1i},x_{20},\ldots,x_{n0})(0\le i\le d)$, 用一元插值算法(如算法 11.1)求出多项式 $P(X_1,x_{20},\ldots,x_{n0})$,
2. $S=\mathrm{skel}P(X_1,x_{20},\ldots,x_{n0})$, 设 $S=\{s_1,s_2,\ldots,s_q\}$,
3. 将 i 顺次由 2 循环到 n, 做如下 4–6 步,
4. 随机任取 d 个值 $x_{i1},x_{i2},\ldots,x_{id}$, 对于每个 $x_{ij}(1\le j\le d)$, 设

$$P(X_1,\ldots,X_{i-1},x_{ij},x_{i+1,0},\ldots,x_{n0})$$

具有如下形式

$$p_1X^{s_1}+p_2X^{s_2}+\cdots+p_qX^{s_q},$$

取 q 组赋值点 $(x_{1k},x_{2k},\ldots,x_{i-1,k})(1\le k\le q)$ 构造独立的线性方程组来求出 $p_1,\ldots,p_q\in F$, 于是得到 d 个多项式

$$P(X_1,\ldots,X_{i-1},x_{ij},x_{i+1,0},\ldots,x_{n0}),(1\le j\le d)$$

5. 利用第 4 步求出的 d 个多项式对每个系数 $p_k(1\le k\le q)$ 进行一元插值算法求出 $p_k(X_i)$, 则

$$P(X_1,\ldots,X_i,x_{i+1,0},\ldots,x_{n0})=p_1(X_i)X^{s_1}+p_2(X_i)X^{s_2}+\cdots+p_q(X_i)X^{s_q},$$

6. $S=\mathrm{skel}P(X_1,\ldots,X_i,x_{i+1,0},\ldots,x_{n0})$, 设 $S=\{s_1,\ldots,s_q\}$,
7. 输出 P.

注152. 在算法第 4 步取 q 组赋值点时, 要使得它们对于要求解的变元 $p_1,\ldots,p_q$ 构成非奇异的独立线性方程组, 否则需要重新选取某些赋值点.

注153. 我们可以选取一组赋值点 $(x_1,x_2,\ldots,x_{i-1})$, 然后令 q 组赋值点为

$$(1,\ldots,1),(x_1^1,\ldots,x_{i-1}^1),(x_1^2,\ldots,x_{i-1}^2),\ldots,(x_1^{q-1},\ldots,x_{i-1}^{q-1}),$$

这样对需求解的 q 个变元来说, 可以得到如下 Vandermonde 线性方程

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ X^{s_1} & X^{s_2} & \cdots & X^{s_q} \\ \vdots & \vdots & & \vdots \\ (X^{s_1})^{q-1} & (X^{s_2})^{q-1} & \cdots & (X_{s_q})^{q-1} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_q \end{pmatrix}$$

$$= \begin{pmatrix} f(1,\ldots,1,x_{ij},x_{i+1,0},\ldots,x_{n0}) \\ f(x_1,\ldots,x_{i-1},x_{ij},x_{i+1,0},\ldots,x_{n0}) \\ \vdots \\ f(x_1^{q-1},\ldots,x_{i-1}^{q-1},x_{ij},x_{i+1,0},\ldots,x_{n0}) \end{pmatrix}.$$

利用此方法的好处是求解线性方程时, Vandermonde 方程其系数矩阵具有特殊性, 因而有特别的线性代数的处理方法(参加 [191]), 求解复杂度低于一般的线性方程. 另一方面, Vandermonde 行列式的非奇异性很容易得到保证, 只需 $X^{s_1},\ldots,X^{s_q}$ 各不相同即可.

对于特征为零的域, 这一点是很容易做到的, 只需将 $x_1,\ldots,x_{i-1}$ 取为前 $i-1$ 个素数即可, 或者当有限域的特征足够大(特征 $p>(2\times3\times\cdots\times p_n)^d$)时也可如此取赋值点, 此时算法仅可能当初始点非精确赋值点时失败, 因此根据定理 11.1 , 我们知道失败的概率

$$\varepsilon<\frac{n(n-1)dT}{2s}.$$

另一方面, 当域的特征 p 有限时, 此时我们随机选取这样 $i-1$ 个赋值点, 当其非精确赋值点或 Vandermonde 行列式奇异时均会失败, 其概率(见 [191]240–242 页)

$$\varepsilon<\frac{n(n-1)dT}{p-1}+\frac{dT(T-1)}{2(p-1)}<\frac{dT(2n^2+T)}{p-1}.$$

如果是有限域 $\mathbb{F}_p$, 并且 $p\le d$ 或者 p 太小不能提供足够多的插值点, 则可以取 $\mathbb{F}_p$ 的扩域, 例如 $\mathbb{F}_{p^k}$, 在其中取插值点来进行计算.

11.2 Euclid 算法和一般模算法

11.2.1 概述

我们已经解决了一元多项式的 GCD 问题, 现在我们考虑多元情形, 即 $\mathbb{Z}[X]=\mathbb{Z}[x_1,\ldots,x_n]$ 中的多项式. 首先根据 Gauss 引理, 我们有:

$$\begin{aligned}\gcd(f,g)&=\operatorname{cont}_{x_n}(\gcd_{x_n}(f,g))\operatorname{pp}_{x_n}(\gcd_{x_n}(f,g))\\&=\gcd(\operatorname{cont}_{x_n}(f),\operatorname{cont}_{x_n}(g))\gcd_{x_n}(\operatorname{pp}_{x_n}(f),\operatorname{pp}_{x_n}(g)).\end{aligned}$$

于是, 我们顺利地将 n 元问题化为了 $n-1$ 元子问题. 将此过程递归地进行, 最终化为一元问题可求解. 显而易见, 这种算法系数的增长是十分迅速的, 不宜采用.

回忆前面处理一元问题采用模算法的思想, 我们希望利用 $\mathbb{Z}[x_1,\dots,x_n]$ 上的模算术来简化问题的计算. 若我们取一个一次多项式 $p=y-a(a\in\mathbb{Z})$, D 为 UFD, 考虑 $R=D[y][x]$ 中的多项式, 并记 R 到 $R/\langle p\rangle$ 的同态像为 $\Phi_p(f)$ 或 $\overline{f}$, 则有下面的定理:

定理11.2. *$f,g\in R$ 均为本原多项式, $h=\gcd(f,g)$, 设 $\overline{h}\neq 0$, 则 $\overline{h}=h_p(=\gcd(\overline{f},\overline{g}))$ 当且仅当 $p=y-a\nmid \operatorname{res}_x(f/h,g/h)$.*

证明. 本定理是定理 8.11 的推广, 证明也与其类似. 首先根据模同态的性质, 我们显然有 $\overline{h}\mid h_p$, 若 $\deg_x h_p=0$, 则 $\overline{h}=h_p$ 显然. 设 $\deg_x h_p\geq 1$, 此是有 s,t 使得 $sf/h+tg/h=\operatorname{res}_x(f/h,g/h)$, 因此 $\overline{s}\overline{f}+\overline{t}\overline{g}=\overline{\operatorname{res}_x(f/h,g/h)}\overline{h}$, 由于 $\overline{\operatorname{res}_x(f/h,g/h)}\neq 0$, 所以 $h_p\mid\overline{h}$, 即 $\overline{h}=h_p$, 充分性得证. 必要性是容易的, 留给读者自行证明. □

于是我们可以得到类似于一元情形的一种赋值同态模算法 [13], 这里不再详细将算法列出, 然而我们需要注意的是这里会有一个领项系数的问题, 例如若两多项式的最大公因子为 $h(x,y)=(y-1)x$, 对 y 进行赋值同态时我们可能会取如下值:

$$y=5,\quad h(x,5)=4x,$$

$$y=7,\quad h(x,7)=6x,$$

这样, 通过两次赋值进行插值即得 $h(x,y)=(y-1)x$, 然而若是我们“不幸”取了一个负值:

$$y=-5,\quad h(x,-5)=6x,$$

注意在 $\mathbb{Z}[x]$ 中 ±1 为可逆元, 因而求其中的 GCD 问题时可相差正负号, 一般情况我们取首项系数为正, 此时将 $h(x,-5)=6x$ 和 $h(x,5)=4x$ 插值则得不到正确的结果, 只有取 $h(x,-5)=-6x$ 时才是正确的. 这个问题仅在最大公因子对主变元 x 的领项系数是一非平凡多项式才会出现, 对于这种情况, 若要比较好地处理, 则须在赋值之前对领项系数进行处理. 后面提到的诸多算法中我们将看到一种领项系数正则化的处理方法.

$\mathbb{F}_p$ 作为一个域较环 $\mathbb{Z}$ 性质更简单, 且其能有效抑制 $\mathbb{Z}$ 上系数膨胀问题. 我们可以综合使用模方法和赋值同态的综合算法来解决题.

11.2.2 $\mathbb{F}_p[x_1,\ldots,x_n]$ 上最大公因子

根据前面的分析, 我们先要解决 $\mathbb{F}_p[x_1,x_2,\ldots,x_n]$ 上求最大公因子的问题, 此要先通过赋值同态将其转化为 $\mathbb{F}_p[x_1]$ 上的问题.

为了说明算法的方便, 本算法中提到的容度 cont, 本原部分 pp, 首项系数 lc 等都是就多项式环 $\mathbb{F}_p[x_n][x_1,\ldots,x_{n-1}]$ 而言的, 亦即 cont 及 lc 都应是 $\mathbb{F}_p[x_n]$ 中的元素.

为了处理领项系数的问题, 算法中采用了正则化的方法. 即对于本原多项式 f,g, 若 $h=\gcd(f,g)$, 则 h 必然也是本原的, 并且 $\mathrm{lc}(h)\mid b=\gcd(\mathrm{lc}(f),\mathrm{lc}(g))$, 因此将模方法求得的 $\overline{h}$ 领项系数化归到 $\overline{b}$ 上再插值还原, 可以得到 $bh/\mathrm{lc}(h)$, 再对其进行本原化处理, 就可以得到结果. 后文中 GCD 算法以及因子分解算法也多次使用了这种正则化方法, 到时将不再说明. 下面将要给出的算法可参考 [74].

算法11.3 (有限域上多元多项式最大公因子算法).

输入:$f,\ g\in\mathbb{F}_p[x_1,\ldots,x_n]$,

输出:$h=\gcd(f,g)$.

1. 若 $n=1$ 则是一元问题, 直接调用相关算法求得首一的 $h=\gcd(f,g)$, 输出 h,
2. 初始化 $a=\gcd(\mathrm{cont}(f),\mathrm{cont}(g))\in\mathbb{F}_p[x_n]$(注意这是一元问题), $f=\mathrm{pp}(f)$, $g=\mathrm{pp}(g)$, $b=\gcd(\mathrm{lc}(f),\mathrm{lc}(g))\in\mathbb{F}_p[x_n]$,
3. 赋初值 $q=1$, $h=1$, $m=\min(\deg_1 f,\deg_1 g)+1$, $vlist=\{\}$,
4. 循环做下面 5–10 步,
5. 随机任取 $v\in\mathbb{F}_p$ 使得 $b(v)\neq 0$ 且 $v\notin vlist$, 将 v 添入 $vlist$ 中,
6. $f_v=f \bmod (x_n-v)$, $g_v=g \bmod (x_n-v)$, 递归调用本算法求解 $n-1$ 元子问题 $h=\gcd(f_v,g_v)$, 令 $m_1=\deg_1 h_v$, $b_v=b(v)$,
7. 将 h_v 正则化使得 $\mathrm{lc}(h_v)=b_v$, 即令 $h_v=b_vh_v/\mathrm{lc}(h_v)$,
8. 若 $m_1<m$ 则令 $q=x_n-v$, $h=h_v$, $m=m_1$,
9. 若上一步不成立且 $m_1=m$, 则利用中国剩余定理或插值算法其出 h, 使得 $h \bmod q$ 不变且 $h \bmod (x_n-v)=h_v$, 即令

$$h=h+\frac{(h_v-h \bmod (x_n-v))q(x_n)}{q(v)},\quad q(x_n)=(x_n-v)q(x_n),$$

10. 若 $\mathrm{lc}(h)=b$ 则:若 $\mathrm{pp}(h)\mid f$ 且 $\mathrm{pp}(h)\mid g$ 则输出 $a\,\mathrm{pp}(h)$, 否则若 $m=0$ 则输出 a.

11.2.3 多元多项式的“Mignotte”界

在处理整系数一元多项式最大公因子时我们曾经引进所谓的 Mignotte 界, 这是 Mignotte 于 1974 年提出的对一元多项式因子系数界的估计. 这一节我们来讨论对于整系数多元多项式同样的问题. 为了后文叙述的方便, 我们先回忆一下对于一元情形的 Mignotte 界, 并表述为如下定理(见定理 8.9):

定理11.3 (Mignotte)**.** 设非零多项式 $f,g\in\mathbb{Z}[x]$ 且 $g\mid f$, $\deg g\leq d$, 则有

$$\|f\|_2\geq 2^{-d}\|g\|_\infty.$$

为了讨论多元情形, 我们给出:

定义11.3 (多元多项式 p-范数)**.** 设有多元多项式

$$f=\sum_{i_1,i_2,\ldots,i_n}a_{i_1,i_2,\ldots,i_n}x_1^{i_1}\cdots x_n^{i_n},$$

则定义其 p-范数为:

$$\|f\|_p=\left(\sum_{i_1,\ldots,i_n}a_{i_1,\ldots,i_n}^p\right)^{1/p}.$$

Coron 于 2004 年提出了二元情形下对 Mignotte 界的推广, 有如下定理(见 [84] 和 [61]3.2 节):

定理11.4 (Coron)**.** 设有非零多项式 $f,g\in\mathbb{Z}[x,y]$, 且 $g\mid f$, $d=\max(\deg_x f,\deg_y f)$, 则有

$$\|f\|_2\geq 2^{-(d+1)^2}\|g\|_\infty.$$

证明. 令 $f^*(x)=f(x,x^{d+1})$, 则有 $\deg f^*\leq(d+1)^2$ 且 $f^*(x)$ 和 $f(x,y)$ 有相同的整系数, 因而 $\|f^*\|_2=\|f\|_2$, 同理令 $g^*(x)=g(x,x^{d+1})$, 则有 $\|g^*\|_\infty=\|g\|_\infty$. 并且有 $g(x,y)\mid f(x,y)$ 可知 $g^*(x)\mid f^*(x)$, 因此根据 Mignotte 界有

$$\|f\|_2\geq 2^{-(d+1)^2}\|g\|_\infty.$$

证毕. □

我们很容易再从二元情况推广到多元情况, 有如下定理:

定理11.5 (多元多项式因子系数界). 设有非零多项式 $f, g \in \mathbb{Z}[x_1, \ldots, x_n]$, 且 $g \mid f$, $d = \max\{\deg_1 f, \ldots, \deg_n f\}$, 则有

$$\|f\|_2 \geq 2^{-(d+1)^n+1}\|g\|_\infty.$$

证明过程是类似的, 只需作相应的替换 $f^*(x) = f(x, x^{d+1}, \ldots, x^{(d+1)^{n-1}})$, 并且注意到 f^* 的次数实际上是不超过

$$d + d(d+1) + \cdots + d(d+1)^{n-1} = d\frac{(d+1)^n - 1}{(d+1) - 1} = (d+1)^n - 1$$

即可.

有了多元多项式的"Mignotte"界, 我们在用模素数方法处理多元多项式时, 至少多了一个用对系数界的估计的方法, 来帮助判断多项式是否能还原到整系数中. 当然, 我们知道 Mignotte 界是对一元情形一个相当好的估计, 本节定理对多元情形的估计虽从其推广而来, 却不一定是最好的估计, 而且其随多项式规模和未定元的个数增长有可能会变得很大, 我们在实际算法中仅将其作为一个参考.

11.2.4 $\mathbb{Z}[x_1, \ldots, x_n]$ 上最大公因子

在我们计算最大公因子的步骤

$$\mathbb{Z}[x_1, \ldots, x_n] \to \mathbb{F}_p[x_1, \ldots, x_n] \to \mathbb{F}_p[x_1] \to \mathbb{F}_p[x_1, \ldots, x_n] \to \mathbb{Z}[x_1, \ldots, x_n]$$

中, 第二环节和第三环节已由算法 11.3 完成, 本节将要讨论首尾两个环节, 如无特别说明, 本节中 cont, pp 等均是就整系数而言的.

算法11.4 (整系数多元多项式最大公因子算法).

输入:$f, g \in \mathbb{Z}[x_1, \ldots, x_n]$,

输出:$h = \gcd(f, g)$.

1. 初始化 $a = \gcd(\mathrm{cont}(f), \mathrm{cont}(g))$, $f = \mathrm{pp}(f)$, $g = \mathrm{pp}(g)$, $b = \gcd(\mathrm{lc}(f), \mathrm{lc}(g))$,
2. 赋初值 $h = 0$, $q = 1$, $m = \min(\deg_n f, \deg_n g)$, $limit = 2^m|b|\min(\|f\|_\infty, \|g\|_\infty)$, $plist = \{\}$,
3. 循环做下面 4 — 9 步,

4. 任取比较大的素数 p 直至 $p \nmid b$ 且 $p \notin plist$,

5. 令 $f_p = f \bmod p$, $g_p = g \bmod p$, $b_p = b \bmod p$, 调用算法 11.3 计算 $h_p = \gcd(f_p, g_p) \in \mathbb{F}_p[x_1, \ldots, x_n]$, 并令 $m_1 = \deg_n h_p$,

6. $h_p = b_p h_p / \operatorname{lc}(h_p)$,

7. 若 $m_1 < m$ 则令 $q = p$, $h = h_p$, $m = m_1$,

8. 若上步判断不成立且 $m_1 = m$, 则用中国剩余定理计算 h 使得 $h \bmod q$ 不变且 $h \bmod p = h_p$, 再令 $q = pq$,

9. 若 $q > limit$ 则:令 $pph = \operatorname{pp}(h)$, 若 $pph \mid f$ 且 $pph \mid g$ 则输出 $apph$, 否则若 $m = 0$ 则输出 a.

关于本算法需要做一点说明. 首先在 m 和 m_1 的计算中都是取了对变元 x_n 的次数 [74], 实际上对所有的变元都取次数来比较也是可行的, 并且可能更准确. 其次, $limit$ 的计算本身并不是多元多项式因子系数的界, 我们可以用前一节的“Mignotte”界来代替, 也可以就用此限制. 这只是一个预判断, 因为当 $q > limit$ 后还有判断. 我们还可以加上一项预判断, 即 h 在中国剩余定理计算前后是否变化, 当不变时再继续后面的算法过程.

11.3 Zippel 稀疏插值算法

Zippel 稀疏算法是求多元多项式最大公因子一个相当有效的算法, 有关这方面的文献可以参见 Zippel 本人的论文及著作 [190], [191]. [14]80–82 页, [74]312–313 页均引用了下面一个具体的例子, 鉴于该算法的理论描述比较复杂, 为了对此算法有一个直观的认识, 我们先看这个具体的例子. 它相当于将稀疏插值一节问题引入中的例子具体化, 取“黑箱”函数

$$h(x, y, z) = F(x, y, z) = \gcd(f(x, y, z), g(x, y, z)).$$

11.3.1 一个具体的例子

例11.1. 求多项式 f,g 的最大公因子 h. 其中

$$\begin{aligned}f =&x^5+2yzx^4+(13yz^2-21y^3z+3)x^3\\&+(26y^2z^3-42y^4z^2+2)x^2+(39yz^2-63y^3z+4yz)x+6,\\g =&x^6+(13yz^2-21y^3z+z+y)x^4+3x^3\\&+(13yz^3+13y^2z^2-21y^3z^2-21y^4z)x^2\\&+(13yz^2-21y^3z+2z+2y)x+2.\end{aligned}$$

解: 首先我们取素数 $p_1=11$, 并得如下关系:

$$h_{11}(x,1,2)\equiv x^3-x+2 \pmod{11},$$

$$h_{11}(x,3,2)\equiv x^3+x+2 \pmod{11},$$

$$h_{11}(x,5,2)\equiv x^3+4x+2 \pmod{11},$$

$$h_{11}(x,-4,2)\equiv x^3+5x+2 \pmod{11},$$

$$h_{11}(x,4,2)\equiv x^3-5x+2 \pmod{11}.$$

用普通的稠密插值算法可以求得 $h_{11}(x,y,2)=x^3+(-3y+2y^3)x+2 \pmod{11}$. 然后我们再对 z 取其它赋值点来计算, 此时我们假定 $h_{11}(x,y,z_i)$ 具有形式 $x^3+(ay+by^3)x+2$, 不必对每个 z 的赋值点 z_i 都取 5 个 y 的赋值点来算, 只需取 2 个点来计算即可. 例如当 $z=-5$ 时, 计算得下面两个式子:

$$h_{11}(x,-3,-5)\equiv x^3-4x+2 \pmod{11},$$

$$h_{11}(x,2,-5)\equiv x^3+5x+2 \pmod{11}.$$

于是我们得到下面的方程组:

$$-3a-5b\equiv -4 \pmod{11},$$

$$2a-3b\equiv 5 \pmod{11},$$

解得 $a=b=-5$, 因此得到 $h_{11}(x,y,-5)=x^3+(-5y-5y^3)x+2 \pmod{11}$. 同理我们还可以得到

$$h_{11}(x,y,-3)\equiv x^3+(-4y-3y^3)x+2 \pmod{11},$$

$$h_{11}(x,y,5)\equiv x^3+(-5y+5y^3)x+2 \pmod{11}.$$

此时利用 z 在 4 个赋值点计算的结果, 利用稠密插值可以得到

$$h_{11}(x,y,z) \equiv x^3 + (2yz^2 + y^3z)x + 2 \pmod{11}.$$

其次, 我们再取素数 $p_2 = 17$, 此时也不必取前面那么多赋值点, 利用稀疏性假设, 我们可以认为 $h_{17}(x,y,z)$ 具有形式 $x^3 + (cyz^2 + dy^3z)x + 2$, 只需取两组 (y,z) 进行插值, 例如:

$$h_{17}(x,7,-4) \equiv x^3 + 8x + 2 \pmod{17},$$

$$h_{17}(x,-2,4) \equiv x^3 + x + 2 \pmod{17},$$

于是解方程可得 $h_{17}(x,y,z) = x^3 + (-4yz^2 - 4y^3z)x + 2 \pmod{17}$. ◇

注154. 利用中国剩余定理将 $\mathbb{F}_{11}$ 和 $\mathbb{F}_{17}$ 中的结果合起来, 即为 $h = x^3 + (13yz^2 - 21y^3z)x + 2$, 经验算, 其恰为所欲求的最大公因子. 这里只用了 13 次一元多项式 GCD 问题求解, 而若用通常的模算法, 则需将近 40 次. 由此可见, 稀疏插值模算法确能有效地减少计算次数.

11.3.2 算法描述

上小节中我们介绍了 Zippel 稀疏插值算法的思想以及具体操作的方法, 现在我们只需稍加改动, 具体给出求值函数的形式, 就可以用来计算最大公因子. 本节中将要给出的算法描述可以参考 [191]15.3 节.

先对记号做一些说明, 在下面将要介绍的插值算法中, 我们保留变元 X_n, 并记之为 Z. 对集合

$$\{(u_1,v_1),\ldots,(u_n,v_n)\}$$

的插值即求多项式 f 使得 $\forall i(1 \le i \le n), f(u_i) = v_i$.

下面的稀疏插值算法 1 和稀疏插值算法 2 两者互相递归调用.

算法11.5 (稀疏插值算法 1).

输入:$l(X_1,\ldots,X_k)$, $f(X_1,\ldots,X_k,Z)$, $g(X_1,\ldots,X_k,Z)$, 其中 $0 \le k < n$,

输出:多项式 f,g 的最大公因子的某个倍数, 使得其关于 Z 的首项系数为 l.

1. 若 $k = 0$ 则输出 $l\gcd(f,g)$,
2. 令 $d = \min(\deg_k f, \deg_k g) + \deg_k l$, 并任取一赋值点 x_k,
3. 递归调用本算法计算 $f(X_1,\ldots,X_{k-1},x_k,Z), g(X_1,\ldots,X_{k-1},x_k,Z)$ 的最大公因子 $I(X_1,\ldots,X_{k-1},Z)$, 其中第一个参数输入 $l(X_1,\ldots,X_{k-1},x_k)$,

4. 令 T 为多项式 I 关于 Z 的系数($\in F[X_1,\dots,X_{k-1}]$)中含最多单项式的系数的单项式个数, 同时对 j 从 0 循环到 D 顺次命 $\mathcal{H}_j=\{(x_k, I\text{关于}Z^j\text{的系数})\}$,
5. 对 i 从 1 循环到 d 顺次做下面 6, 7 步,
6. 随机取不重复的赋值点 y_i, 调用算法 11.6 , 输入 skel$I$, $l(X_1,\dots,X_{k-1},y_i)$, $F(X_1,\dots,X_{k-1},y_i,Z)$, $G(X_1,\dots,X_{k-1},y_i,Z)$, T, 得到稀疏插值求得的多项式 W,
7. 对 j 从 0 循环到 D 顺次命 $\mathcal{H}_j=\mathcal{H}_j\cup\{(y_i, W\text{关于}Z^j\text{的系数})\}$,
8. 对 j 从 0 循环到 D 顺次利用稠密插值算法由 $\mathcal{H}_j$ 计算到到多项式 $h_j\in F[X_1,\dots,X_k]$, 令 $h=h_DZ^D+\cdots+h_0$,
9. 若 h 不能同时整除 f 和 g 则退出整个算法重新选取赋值点(整个算法失败), 否则输出 h.

算法11.6 (稀疏插值算法 2)**.**

输入:模板 S, 多项式 $l(X_1,\dots,X_k)$, $f(X_1,\dots,X_k,Z)$, $g(X_1,\dots,X_k,Z)$, T,

输出:由它们计算得到稀疏插值的结果 h, 也即 f, g 的最大公因子的某个倍数, 其关于 Z 的首项系数为 l.

1. 若 $k=0$ 则输出 $l\gcd(f,g)$,
2. 任取 k 个赋值点 $y_1,\dots,y_k$,
3. 对 i 从 1 循环到 T 顺次做下面 4, 5 步,
4. 令 $W=l(y_1^i,\dots,y_k^i)\times\gcd(f(y_1^i,\dots,y_k^i,Z),g(y_1^i,\dots,y_k^i,Z))$,
5. 对 j 从 0 循环到 D 顺次命 $\mathcal{H}_j=\mathcal{H}_j\cup\{W\text{关于}Z^j\text{的系数}\}$,
6. 对 j 从 0 循环到 D, 由 $\mathcal{H}_j$, 以及 S 中含 Z^j 的项和 $y_1,\dots,y_k$ 计算出 Vandermonde 矩阵各元素, 解线性方程组可得 h_j(具体方法见注 156),
7. 输出 $h_DZ^D+\cdots+h_0$.

注155. 算法 11.5 第 1 步, 算法 11.6 第 1 步, 第 4 步等计算一元 GCD 时均是指求得其首一化的 GCD. 其 $l\gcd(f,g)$ 时, 可以先乘上 l 再除以原先多项式的领项系

数, 避免出现分数.

注156. 算法 11.6 第 6 步计算的具体方法是:令 S_j 为输入模板 S 中最后一项指标为 j(即 Z 的次数为 j)的元素在前 k 维上投影的集合. 例如对于模板

$$\{(1,1),(1,0),(0,0)\}$$

有 $S_0=\{(1),(0)\}$, $S_1=\{(1)\}$. 设 S_j 的元素个数为 s_j, 由赋值点 $y_1,\ldots,y_k$ 和指标集 S_j 可以计算得到一个 s_j 阶的 Vandermonde 矩阵(若此矩阵奇异, 则返回算法第 2 步重新选取赋值点). 再取 $\mathcal{H}_j$ 的前 s_j(肯定小于 T)项, 以此可得一线性方程组. 其解配上相应的模板 S_j 即到多项式 h_j.

如果 f 和 g 都是关于 Z 的首一多项式, 则直接调用算法 11.5 , 输入参数 $1,f,g$ 即可, 但是对于关于 Z 的首项系数不为 1 的多项式, 我们可如下给出它们的最大公因子. 其中为了使我们每一步最后的试除法能够成功, 调用算法 11.5 时我们输入 l, lf, lg. 此时, 因为要使稠密插值结果的首项系数为 l, 所以我们在算法 11.5 第 2 步中令 $d=\min(\deg_k f,\deg_k g)+\deg_k l$.

算法11.7 (稀疏插值算法).

输入:多项式 $f,g\in F[X_1,\ldots,X_n]$,

输出:f,g 的最大公因子.

1. 递归调用本算法计算 $f_1=\mathrm{cont}_Z(f)$, $g_1=\mathrm{cont}_Z(g)$, $a=\gcd(f_1,g_1)$, 并令 $f=f/f1$, $g=g/g1$,
2. 递归调用本算法计算 $l=\gcd(\mathrm{lc}_Z(f),\mathrm{lc}_Z(g))$, 调用算法 11.5 , 输入参数 l,lf,lg, 得到 h,
3. 输出 $a\mathrm{pp}(h)$.

11.4 求 GCD 的其它方法

关于多元多项式的 GCD 问题, 前述的一般算法以及 Zippel 稀疏插值模算法已经能够相当有效地处理, 限于篇幅, 这一节所说的方法仅是作一些补充说明, 算法的细节不再详细说明, 有兴趣可以参考列出的一些文献.

11.4.1 启发式算法(Heuristic GCD)

启发式算法是一种将多元多项式 GCD 问题转化为大整数问题的算法. 此算法的优势在于它将问题化为整数计算问题, 若能够高效处理整数运算则可以提高计算效率. 其次, 对于一般的小规模问题来说, 用启发式算法还是比插值算法可能来得快. 读者可以参考 [13]. 至于算法的具体实现, 可参考 [74]7.7 节.

11.4.2 EZ-GCD

EZ-GCD算法即 Extended Zassenhaus 算法(参考 [130]), 它利用 Hensel 提升来计算多元多项式的 GCD. 此算法的具体实现可以参考 [74]7.6 节.

11.5 多元多项式因子分解的 Kronecker 算法

由多元 Mignotte 界一节我们很容易想到用 x 的幂次代替其它变元的同态方法, 即 Kronecker 算法.

对于给定的多项式 $f \in \mathbb{Z}[x_1, x_2, \ldots, x_n]$, 我们取主变元为 x_1, 并记之为 x. 设

$$d = \max_{1 \le i \le n} \deg_{x_i} f,$$

如果我们取多项式

$$\tilde{f}(x) = f(x, x^{d+1}, x^{(d+1)^2}, \ldots, x^{(d+1)^{n-1}}),$$

易知如果有不可约因子分解 $f = f_1 f_2 \cdots f_r$, 这里我们假设 f 关于 x 是无平方因子本原多项式, 那么有

$$\tilde{f} = \tilde{f}_1 \cdots \tilde{f}_r.$$

然而当我们对 $\tilde{f}$ 进行因子分解时, 得到的不一定是上式, 因为 $\tilde{f}_i$ 不一定均是不可约的. 设 $\tilde{f}$ 的不可约因子分解为

$$\tilde{f} = g_1 g_2 \cdots g_t,$$

则由因子组合算法可以还原在 $\mathbb{Z}[x_1, \ldots, x_n]$ 中的分解.

由此可见, Kronecker 算法的思想本身是简单的, 下面给出具体的算法.

算法11.8 (Kronecker 因子分解算法).

输入:关于 $x = x_1$ 本原且无平方因子的多项式 $f \in \mathbb{Z}[x_1, \ldots, x_n]$,

输出:其各不可约因子 $\{f_1, \ldots, f_r\}$.

1. 令 d 为各变元次数的上界, 求得多项式

$$\tilde{f}(x) = f(x, x^{d+1}, x^{(d+1)^2}, \ldots, x^{(d+1)^{n-1}})$$

的因子分解

$$\tilde{f} = g_1 g_2 \cdots g_t.$$

2. 令 $T = \{1, 2, \ldots, t\}$, $s = 1$, $result = \{\}$, $h = f$,

3. 若 $2s \leq \#T$, 则循环做下面 4–6 步, 否则转第 7 步,

4. 枚举 T 的所有 s 元子集 S, 并做下面第 5 步,

5. 由多项式 $\prod_{i \in S} g_i \in \mathbb{Z}[x]$ 我们可以还原得到多项式 $g \in \mathbb{Z}[x_1, \ldots, x_n]$(见注 157), 若 $g \mid h$ 则令 $result = result \cup \{g\}$, $h = h/g$, $T = T \setminus S$, 并转第 3 步,

6. $s = s + 1$, 转第 3 步,

7. 输出 $result \cup \{h\}$.

注157. 由某一多项式 $\tilde{f}(x)$ 还原得到多项式 $f(x_1, \ldots, x_n)$ 的方法是:对 $\tilde{f}$ 中的任何一个单项式 ax^b, 由连续对 $d+1$ 的除法可以得到 n 元序对 $(b_1, \ldots, b_n)$, 其中 $0 \leq b_i \leq d$ 使得

$$b = b_1 + b_2(d+1) + b_3(d+1)^2 + \cdots + b_n(d+1)^{n-1},$$

将此单项用 $ax_1^{b_1} x_2^{b_2} \cdots x_n^{b_n}$ 代替即可还原多项式.

11.6 利用 Hensel 提升的因子分解算法

11.6.1 概述

类似于多元 GCD 求解, 利用赋值同态的方法, 我们也可以将多元因子分解问题转化为一元问题. 我们很容易会产生如下一般性的想法, 这里假设我们考虑 $f \in \mathbb{Z}[x_1, x_2, \ldots, x_n]$ 的分解, 并将变元 x_1 视为主变元 x.

- 将 f 化为关于主变元 x 本原以及无平方因子的多项式. 这一点是很容易做到的, 由多元 GCD 算法可以求出 f 关于 x 各系数多项式的最大公因子, 而由 $\gcd(f, \partial f/\partial x)$ 可以将其化为无平方因子多项式的分解问题.

- 利用 $I=\langle x_2-a_2,\ldots,x_n-a_n\rangle$ 对应的赋值同态将 f 化为 $\tilde{f}=f \bmod I$, 即 $\tilde{f}(x)=f(x,a_2,\ldots,a_n)$, 使得 $\tilde{f}$ 无平方因子并且 $\deg_x \tilde{f}=\deg_x f$.
- 处理一元分解问题, 得到不可约分解 $\tilde{f}=g_1g_2\cdots g_r$, 这里可以将 $\mathrm{cont}\tilde{f}$ 任意分配到不可约因子 g_i 上.
- 利用类似于 Hensel 提升的方法, 将模 I 下的分解提升到足够高的模 I^k 下的分解, 得到多元因子, 最后采取诸如因子组合的方法将其还原到整系数多项式中的因子.

关于赋值点和主变元的选取, 这里做一些补充. 赋值点应优先选择 $\pm1,0$, 以保证 $\tilde{f}$ 无平方因子且关于 x 次数不变. 选择 $\pm1,0$ 的好处是使得系数较小, 由后面的算法我们看出对于非零赋值点, 我们将对变量做一平移, 以使提升算法的模运算也便于进行. 其次还要考虑使得 $\tilde{f}$ 的不可约因子数尽可能少, $\tilde{f}$ 关于 x 的首项系数尽可能小, 如果是 1 更好, 此时所有的不可约因子的首项系数都将为 1.

11.6.2 扩展 Zassenhaus 算法

现在我们要介绍的扩展 Zassenhaus 算法([178] 第 8 节)解决如下问题. 设 $\tilde{f}\in Z[x]$ 有分解 $\tilde{f}=gh$, 其中 $\gcd(g,h)=1$, 亦即

$$f\equiv gh \pmod I,$$

现在需找到 g_k, h_k 使得

$$f\equiv g_kh_k \pmod{I^k},\quad f_k\equiv f \pmod I,\quad g_k\equiv g \pmod I.$$

对于 $i=2,3,\ldots,n$, 记 $y_i=x_i-a_i$, 则此时 $I=\langle y_2,\ldots,y_n\rangle$, 并记 $f^*(x_1,y_2,\ldots,y_n)=f(x_1,y_2+a_2,\ldots,y_n+a_n)$. 基于这些符号上的说明, 我们可以提出一种归纳的算法如下:

算法11.9 (EZ 算法).

输入:多项式 f^*(即 f), g_{k-1}, h_{k-1}, 并且满足前述相应条件,

输出:提升后的 g_k, h_k.

1. 计算 r_{k-1}, e_{k-1} 使得

$$r_{k-1}=g_{k-1}h_{k-1}-f^*,\quad e_{k-1}\equiv r_{k-1} \pmod{I^k},$$

2. 利用扩展 Euclid 算法计算唯一的 $\alpha_i(x)$, $\beta_i(x)$ 使得

$$\alpha_i(x)g(x)+\beta_i(x)h(x)=x^i,$$

并且 $\deg\alpha_i<\deg h$. (命 $\alpha_i=\alpha_0x^i \bmod h$, 即可. 此时若 $i<\deg g+\deg h$ 则有 $\deg\beta_i<\deg g$. 并且此步可略去, 因为可以在整个算法开始提升的第一步计算足够多的 α_i, β_i, 保存起来并供后面各步提升时使用.)

3. 将 e_{k-1} 中的 x^i 用 $\alpha_i(x)$ 代替得到多项式 $A(e_{k-1})$, 同理将 e_{k-1} 中的 x^i 用 $\beta_i(x)$ 代替得到多项式 $B(e_{k-1})$.

4. 令 $g_k=g_{k-1}-B(e_{k-1})$, $h_k=h_{k-1}-A(e_{k-1})$ 并输出 g_k, h_k.

算法有效性. 首先由 $A(e_{k-1})$ 和 $B(e_{k-1})$ 的定义可知有

$$A(e_{k-1})g(x)+B(e_{k-1})h(x)=e_{k-1}$$

成立. 由 $e_{k-1} \bmod I^{k-1}=r_{k-1} \bmod I^{k-1}=0$ 知 e_{k-1} 是 $y_2,y_3,\dots,y_n$ 的 $k-1$ 次齐次多项式. 因而 $A(e_{k-1})$ 和 $B(e_{k-1})$ 也是它们的 $k-1$ 次齐次式. 由

$$g_{k-1}=g-B(e_1)-B(e_2)-\cdots-B(e_{k-2})$$

可知

$$A(e_{k-1})g_{k-1}(x)\equiv A(e_{k-1})g(x)\pmod{I^k},$$

同理有

$$B(e_{k-1})h_{k-1}(x)\equiv B(e_{k-1})h(x)\pmod{I^k},$$

因而

$$r_{k-1}\equiv e_{k-1}\equiv A(e_{k-1})g(x)+B(e_{k-1})h(x)\pmod{I^k}.$$

并且我们有 $A(e_{k-1})B(e_{k-1})\equiv 0 \pmod{I^k}$. 命 $r_k=g_kh_k-f^*$, 则

$$r_k\equiv r_{k-1}-A(e_{k-1})g_{k-1}-B(e_{k-1})h_{k-1}+A(e_{k-1})B(e_{k-1})\equiv 0\pmod{I^k}.$$

亦即 $f^*\equiv g_kh_k \pmod{I^k}$. □

在向 I^k 提升的过程中, 倘若有某一步 $r_l=0$, 则之后无须提升.

我们在利用算法 11.9 处理 $\mathbb{Z}[x]$ 中的多项式时, 实际上已经是在 $\mathbb{Q}[x]$ 中进行运算了, 这一点可以从算法 11.9 第 2 步中要计算 Bezout 系数可以看出来. 在最后进行因子还原时, 由于领项系数的处理可以使得求出的因子仍然是在 $\mathbb{Z}[x]$ 中. 本

章开始曾提到我们不仅利用赋值同态, 还可以利用模同态来简化计算, 那么扩展 Zassenhaus 算法还适用吗?事实上, 只要成立 Bezout 等式(未必为 Euclid 整环)即可. 如果多项式的系数在模环 $\mathbb{Z}_m$ 中, 其中 m 为一素数幂 p^l, 那么条件是满足的, 下面我们来说明这一事实.

引理11.1. *$a \in \mathbb{Z}_{p^l}$ 可逆当且仅当 $p \nmid a$.*

证明. 充分性:若 $p \nmid a$, 则 $\gcd(a, p^l) = 1$, 由 Bezout 等式可知 a 在 $\mathbb{Z}_{p^l}$ 中可逆.

必要性:若 $p \mid a$, 则在 $\mathbb{Z}_{p^l}$ 中有 $p^{l-1} \cdot a = 0$, 从而 a 不可逆. □

下面给出的定理揭示了 $\mathbb{Z}_{p^l}[x]$ 类似于 Euclid 整环的性质.

定理11.6. *设 $g, h \in \mathbb{Z}_{p^l}[x]$ 满足 $p \nmid \mathrm{lc}(g)$, $p \nmid \mathrm{lc}(h)$, 且 $\gcd(\Phi_p(g), \Phi_p(h)) = 1$, 则 $\forall f \in \mathbb{Z}_{p^l}[x]$ 都存在唯一的多项式 $s, t \in \mathbb{Z}_{p^l}[x]$ 使得 $sg + th \equiv f \pmod{p^l}$, 且 $\deg s < \deg h$. 若 $\deg f < \deg g + \deg h$, 还有 $\deg t < \deg g$.*

证明. 存在性:首先由 $\mathbb{F}_p[x]$ 是 Euclid 整环我们知道 $\exists s^{(1)}, t^{(1)} \in \mathbb{F}_p[x]$ 使得

$$s^{(1)}g + t^{(1)}h \equiv 1 \pmod{p}.$$

设 $s^{(k)}, t^{(k)}$ 已求得并使 $s^{(k)}g + t^{(k)}h \equiv 1 \pmod{p^k}$, 则定义迭代算法如下:

设 $s_k, t_k \in \mathbb{F}_p[x]$ 是

$$s_k g + t_k h \equiv \frac{1 - s^{(k)}g - t^{(k)}h}{p^k} \pmod{p}$$

的解, 再令

$$s^{(k+1)} = s^{(k)} + s_k p^k, \quad t^{(k+1)} = t^{(k)} + t_k p^k.$$

显然

$$s^{(k+1)}g + t^{(k+1)}h = s^{(k)}g + t^{(k)}h + p^k(s_k g + t_k h) \equiv 1 \pmod{p^{k+1}}.$$

因此 $fs^{(l)}g + ft^{(l)}h \equiv f \pmod{p^l}$. 由题设条件和引理 11.1 知 $\mathrm{lc}(h)$ 是 $\mathbb{Z}_{p^l}$ 中可逆元, 我们可以作除法:

$$fs^{(l)} = uh + s \pmod{p^l},$$

其中 $\deg s < \deg h$. 再定义 $t = ft^{(l)} + ug$, 即知 s, t 满足定理.

唯一性:由 $s^{(l)}, t^{(l)}$ 的存在知道在 $\mathbb{Z}_{p^l}$ 中 g 和 h 也互素, 设满足定理的还有 s', t', 则易得 $(s - s')g = (t' - t)h$, 由 $\deg(s - s') < \deg h$ 且 $h \mid (s - s')$ 知 $s - s' = t - t' = 0$. □

于是, 在 $\mathbb{Z}_{p^l}$ 中我们可以用扩展 Zassenhaus 算法, 利用该算法之前, 我们先取小素数 p 计算 $\mathbb{F}_p$ 中 $\tilde{f}=\Phi_I(f)$ 的分解, 再利用 Hensel 提升算法(Zassenhaus 算法)可以得到进行扩展 Zassenhaus 算法需要输入的分解.

下面给出一个利用扩展 Zassenhaus 算法求因子分解的简单的例子.

例11.2. 设 $f=x^2-3xz^2+2x-yx+3yz^2-2y+zx-3z^3+2z$. 取 $p=7$, $l=1$, $I=\langle y,z\rangle$, 有

$$f\equiv x(x+2)\pmod{I,p^l},$$

记 $g=g_1=x$, $h=h_1=x+2$, 因为在 $\mathbb{Z}_{p^l}[x]$ 中有 $3g_1-3h_1\equiv 1\pmod{p^l}$, 即 $\alpha_0=3,\beta_0=-3$, 这里预先将各个 α_i,β_i 计算好, 可得

$$\alpha_1=x\alpha_0 \bmod h=1,$$

$$\beta_1=x\beta_1 \bmod g=0.$$

第一步提升, 计算得

$$e_1\equiv g_1h_1-f\equiv (x+2)y-(x+2)z\pmod{I^2,p^l},$$

因此

$$A(e_1)=(y-z)\times 1+2(y-z)\times 3=0,$$

$$B(e_1)=(y-z)\times 0+2(y-z)\times(-3)=y-z,$$

于是 $g_2=g_1-B(e_1)=x-y+z$, $h_2=h_1-A(e_1)=x+2$.

再一次提升计算, 可得

$$e_2\equiv g_2h_2-f\equiv 3z^2x\pmod{I^3,p^l},$$

因此

$$g_3=g_2-B(e_2)=g_2-3z^2\times 0=x-y+z,$$

$$h_3=h_2-A(e_2)=h_2-3z^2\times 1=x+2-3z^2,$$

此时已有 $f=g_3h_3$.

11.6.3 因子还原

不妨设我们已得到因子分解

$$f\equiv g_1g_2\cdots g_t\pmod{m,I^k},$$

其中模去一个整数 m, 是指在计算提升之前可以选择模掉一足够大的 m 进行运算, 以减小系数膨胀. 当然也可以直接在整数环中计算, 此种情况我们统一用 $m = 0$ 表示.

因子还原的过程也是一个因子组合算法, 这一算法已经多次出现, 因此这里直接给出还原的算法.

算法11.10 (EZ 算法结果的因子还原).

输入:已知分解 $f \equiv g_1 \cdots g_t \pmod{m, I^k}$,

输出:f 的不可约因子集合 $\{f_1, f_2, \ldots, f_r\}$.

1. 令 $T = \{1, 2, \ldots, t\}$, $s = 1$, $result = \{\}$, $h = f$, $h^* = \mathrm{lc}(h)h$,
2. 若 $2s \leq \#T$, 则循环做下面 3–5 步, 否则转第 6 步,
3. 枚举 T 的所有 s 元子集 S, 并做下面第 4 步,
4. $g = \prod_{i \in S} g_i \bmod (m, I^k)$, $g^* = \mathrm{lc}(h)\,\mathrm{lc}(g)^{-1} g \bmod (m, I^k)$, 若 $g^* \mid h^*$ 则令 $result = result \cup \{\mathrm{pp}(g^*)\}$, $h = h/\mathrm{pp}(g^*)$, $h^* = \mathrm{lc}(h)h$, $T = T \setminus S$, 并转第 2 步,
5. $s = s + 1$, 转第 2 步,
6. 输出 $result \cup \{h\}$.

11.6.4 预先确定因子的首项系数

前几节组合起来可以给出一个完整的因子分解算法, 然而其仍有一些效率上的不足. 其中因子还原之后我们要将在模 I^k 中的结果回复到正常的多项式, 这涉及到 Taylor 展开, 所以我们前面尽量选择较小的赋值点, 以减轻此处计算形如 $(x-u)^k$ 展开的负担. 另外, 首项系数的不确定使得 EZ 算法提升时的中间多项式将会比较复杂, 我们可以看下面一个因子分解的例子.

例11.3. 考虑多项式 $f = (y+2)^2 x^2 - 1$ 的因子分解.

解: 这里取 x 为主变元, y 取赋值点 0, 则有分解 $f \equiv (2x+1)(2x-1) \pmod{y}$. 设 $g = 2x+1$, $h = 2x-1$, 提升算法给出

$$g_2 = \frac{1}{2}(2 + y + 4x(1+y)), \quad h_2 = \frac{1}{2}(-2 + 4x + y),$$

$$g_3 = \frac{1}{2}(2+y)(1+x(2+y)), \quad h_3 = \frac{1}{4}(-4+8x+2y-y^2).$$

这时再经过一次首项系数的处理并取本原部分方可得到真正的因子 $(1+x(2+y))$ 和 $(-1+x(2+y))$. ◇

对于上例而言, 若我们能预先确定各因子的首项系数, 在提升之前就将 g, h 中的首项系数代替为实际首项系数, 即 $g=(2+y)x+1$, $h=(2+y)x-1$, 则提升算法大大简化(对于此特例恰好无需任何提升).

Paul S. Wang[177] 于 1978 年改进原先的 EZ 算法, 提出了在提升之前预先确定因子首项系数多项式的方法(在 [177] 中, Wang 还提出了改进的 EZ 算法, 即 EEZ 算法, 有兴趣的读者可见该文第 5, 6, 7 节), 下面我们就来介绍这一方法.

设多项式 f 关于主变元的首项系数为 $J=\mathrm{lc}(f,x)\in\mathbb{Z}[x_2,\ldots,x_n]$, 设其不可约因子分解为

$$J=\Omega\prod_{1\le i\le k}J_i^{e_i},$$

其中 $\Omega=\mathrm{cont}_{\mathbb{Z}}(J)$, $J_i(1\le i\le k)$ 为其各不可约因子. 此时对于 $\tilde{f}=f \bmod I$ 有分解

$$\tilde{f}=\delta g_1 g_2\cdots g_r,$$

其中 $\delta=\mathrm{cont}\tilde{f}$. 我们的目的即是要确定各 g_i 的首项系数, 并将 δ 合理地分配到各个 g_i 上. 我们在选择赋值点时, 需要满足如下几个条件:

定理11.7. *选取赋值点 $a_2,a_3,\ldots,a_n$, $\tilde{f}=f(x,a_2,\ldots,a_n)$ 使得:*

1. $\deg\tilde{f}=\deg f$, *即* $J(a_2,\ldots,a_n)\neq 0$,
2. $\tilde{f}$ *无平方因子, 即* $\mathrm{res}_x(f,\partial f/\partial x)(a_2,\ldots,a_n)\neq 0$,
3. *对任意* J_i, $\tilde{J}_i=J_i(a_2,\ldots,a_n)$ *至少一有个素因子* p_i, *其不能整除* $J_j(\forall j<i)$ *和* Ω, δ.

这样的赋值点有无穷多组.

显然满足前两个条件的有无穷多组, 若要说明满足第三个条件的点有无穷多组, 可见 [177]1218 页的说明, 这里不再花篇幅叙述了.

选取赋值点时要满足前两个条件的原因我们在前文已经论述过了, 而第三个条件看上去不那么直观, 实际上, 它是为了保证各因子首项系数能够正确地定下来. 设 $g_1,\cdots,g_r$ 中没有无关的因子, 即它们都对应了 f 分解的得到的 r 个因子, 无须组合就能还原, 设 f 有分解

$$f=f_1 f_2\cdots f_r,$$

令 $C_i = \mathrm{lc}(f_i, x)$, $\tilde{C}_i = C_i(a_2, \ldots, a_n)$, 且

$$\tilde{f}_i = f_i(x, a_2, \ldots, a_n) = \delta_i g_i,$$

其中 $\prod_{1\le i\le r} \delta_i = \delta$. 那么, 我们有如下的引理:

引理11.2. 对任何 $i(1 \le i \le r)$ 和 $m \in \mathbb{N}$, 我们有 $J_k^m \mid C_i$ 当且仅当 $\tilde{J}_k{}^m \mid \mathrm{lc}(g_i)\delta$.

证明. 若 $J_k^m \mid C_i$, 则 $\tilde{J}_k{}^m \mid \tilde{C}_i = \mathrm{lc}(\tilde{f}_i, x) = \mathrm{lc}(g_i)\delta_i \mid \mathrm{lc}(g_i)\delta$.

若 $J_k^m \nmid C_i$, 可设 $\tilde{C}_i = \tilde{J}_1{}^{s_1} \cdots \tilde{J}_k{}^{s_k}\omega$, 其中 $\omega \mid \Omega$, $s_i \ge 0$ 且 $s_k < m$. 于是 $p_k^m \nmid \tilde{C}_i$, 故 $\tilde{J}_k{}^m \nmid \mathrm{lc}(g_i)\delta$. □

由引理 11.2 , 我们可以从 J_k 开始分配, 一直将 J_1 分配完, 这样得到各个 C_i 的本原部分 $D_i = \mathrm{pp}(C_i)$. 设记号都同前, 我们将这一算法叙述如下:

算法11.11 (分配首项系数算法 1).

计算 $D_i(1 \le i \le r)$.

1. 令各个 $D_i = 1$,
2. 命 j 由 k 递减到 1, 依次执行第 3 步,
3. 求出 m_i 使得 $\tilde{J}_j{}^{m_i} \mid \mathrm{lc}(g_i)\delta$ 且 $\tilde{J}_j{}^{m_i+1} \nmid \mathrm{lc}(g_i)\delta$, 命 $D_i = D_i J_i^{m_i}$,
4. 输出各个 D_i.

经过这一算法后, J 的各个非平凡因子已经分配完, 只余下整数容度 Ω 没有分配. 此时有

$$\Omega D_1 D_2 \cdots D_r = \delta \,\mathrm{lc}(g_1)\,\mathrm{lc}(g_2)\cdots\mathrm{lc}(g_r).$$

若要得到最终结果 C_i, 则仍需将 Ω 和 δ 合理分配到诸因子上. 如果 $\delta = 1$, 则这一过程很简单, 只要作一些正则化的手续即可, 即命

$$C_i = \frac{\mathrm{lc}(g_i)}{\tilde{D}_i} D_i,$$

这样, 各个 C_i 就能和分解得到的 g_i 对应上.

但若 $\delta \ne 1$, 则这一过程稍微复杂一些, 我们用下面的算法来求得 C_i:

算法11.12 (分配首项系数算法 2).

1. 令 $d=\gcd(\mathrm{lc}(g_i),\tilde{D}_i)$, $C_i=\mathrm{lc}(g_i)D_i/d$,
2. 令 $g_i=\tilde{D}_i g_i/d$,
3. 令 $\delta=\delta/(\tilde{D}_i/d)$.

算法 11.12 为了将 D_i 对应到相应的 g_i 上, 实际上是将 $\tilde{D}_i$ 和 $\mathrm{lc}(g_i)$ 均对应到它们的最小公倍数上去, 对应过程中, D_i 将 Ω 分配, g_i 将 δ 分配.

经过算法 11.12 后, 一般情况下 δ 应该为 1. 如果不为 1, 说明 f 不是本原多项式. 此时令 $C_i=\delta C_i$, $g_i=\delta g_i$, $f=\delta^{r-1}f$ 即可. 进行 Hensel 提升之前, 将各个 g_i 的领项系数换为 C_i, 后续算法的计算将会得到简化.

现在, 我们回到定理 11.7 的第三个条件上来. 上文我们已经解释了第三个条件的作用以及怎样利用它完成各因子首项系数的确定, 然而如何选取求值点才能满足第三个条件呢?若按照条件所叙述的来进行检测, 势必要进行很多整数的因子分解过程, 这样不一定划算, 下面我们给出一个算法, 只需要用到整数的最大公因子计算.

算法11.13 (分配首项系数算法 3).

输入:Ω, $\tilde{J}_i$, δ,

输出:定理 11.7 第三个条件是否满足以及整数序列 $d_i(1\leq i\leq k)$, 其中 d_i 的任一素因子均可视作满足条件的 p_i.

1. 命 $d_0=\delta\Omega$,
2. 命 i 由 1 递增到 k, 执行 3–8 步,
3. 命 $q=|\tilde{J}_i|$,
4. 命 j 由 $i-1$ 递减到 0, 执行 5–7 步,
5. 命 $r=d_j$,
6. 命 $r=\gcd(r,q)$, $q=q/r$, 若 $r\neq 1$, 则一直执行本步,
7. 若 $q=1$, 返回条件不满足,

8. 命 $d_i = q$,

9. 返回条件满足和 $d_1, d_2, \ldots, d_k$.

注158. 由于 d_i 中任一素因子均不整除其它的 $\tilde{J}_j (j \neq i)$, 因此在算法 11.11 第三步通过试除得到 m_i 时完全可以用 d_i 代替 $\tilde{J}_i$ 进行试除.

12 一元多项式求根算法

一元多项式求根是计算机代数中一个重要的问题, 同时, 它也是代数方程组求解以及代数数表示和运算的基础. 本章将围绕这一问题, 从数值解和符号解两个方面进行阐述.

一元多项式方程的数值求根所采用的方法同大部分数值算法一样, 都是用迭代的方法. 对于一般的方程 $f(x)=0$ 的求解(f 可导), 我们熟知有 Newton 迭代算法:

$$x_{\lambda+1}=x_\lambda-\frac{f(x_\lambda)}{f'(x_\lambda)},$$

多项式求根也可采用此迭代算法, 并且其收敛阶数可以利用 Taylor 公式进行估计. 设 x 为方程的一根, 由于

$$\begin{aligned}f(x_\lambda)&=f(x)+f'(x)(x_\lambda-x)+O((x_\lambda-x)^2)\\&=(f'(x_\lambda)+f''(x_\lambda)(x-x_\lambda)+O((x_\lambda-x)^2))(x_\lambda-x)+O((x_\lambda-x)^2)\\&=f'(x_\lambda)(x_\lambda-x)+O((x_\lambda-x)^2),\end{aligned}$$

我们有

$$x_{\lambda+1}-x=x_\lambda-x-\frac{f(x_\lambda)}{f'(x_\lambda)}=\frac{O((x_\lambda-x)^2)}{f'(x_\lambda)},$$

当 $f'(x)\neq 0$, 即 x 是单重根时, 由上式可知迭代是二阶收敛的. 若 $f'(x)=0$, 即 x 是多重根时, 我们有

$$\begin{aligned} x_{\lambda+1}-x &= x_\lambda - x - \frac{f(x_\lambda)}{f'(x_\lambda)} \\ &= x_\lambda - x - \frac{O((x_\lambda-x)^2)}{O(x_\lambda-x)} \\ &= O(x_\lambda - x), \end{aligned}$$

可知其为一阶线性收敛的.

本章我们将着重介绍 Jenkins-Traub 数值求根算法, 简单介绍 Laguerre 等算法.

关于一元多项式方程的符号求根, 大致上我们采取这样一种思路:将复杂的多项式通过函数复合的解和因子分解化为较为简单的可精确求解的多项式. 目前我们已知的可精确求解的多项式有不高于四次的多项式以及分圆多项式. 对于前者, 我们利用求根公式可以将解用根式表达出来, 对于后者利用 1 的单位根也可以表示出来.

在本章中间, 我们也会插入有限域上多项式求根以及根的区间隔离算法. 区间隔离算法可以帮助数值算法的迭代初始点的选取, 并且可用于多元多项式组求根以及代数数的运算.

12.1 多项式零点模估计

在后面的数值算法以及区间隔离等算法中, 我们均需要对多项式零点的模估计其上界或下界, 因此本节介绍若干对零点模的估计. 首先下面的 Cauchy 不等式给出了 $\mathbb{C}[x]$ 上多项式根的模的一个估计.

定理12.1. 设 $f=\sum_{0\leq i\leq n} f_i x^i\in\mathbb{C}[x]$, $r_1,\ldots,r_n$ 是它的根, 则对任何一个根 r 均有

$$r<1+\frac{\max(|f_0|,|f_1|,\ldots,|f_{n-1}|)}{|f_n|}.$$

证明. 若 $|r|\leq 1$, 则无需证明. 下面假设 $|r|>1$, 由 $\sum_{0\leq i\leq n} f_i r^i=0$ 可得

$$\begin{aligned} |f_n r^n| &= \left|\sum_{0\leq i\leq n-1} f_i r^i\right| \leq \max_{0\leq i\leq n-1}|f_i|\cdot(1+|r|+\cdots+|r|^{n-1}) \\ &= \max_{0\leq i\leq n-1}|f_i|\cdot\frac{|r|^n-1}{|r|-1} < \max_{0\leq i\leq n-1}|f_i|\frac{|r|^n}{|r|-1}, \end{aligned}$$

因此得到估计 $|r|<1+\dfrac{\max_{0\leq i\leq n-1}|f_i|}{|f_n|}$. □

考虑多项式 $r^n f(1/r)$ 很容易得到:

推论12.1. 各记号同定理 *12.1* , 对任何 f 的根 r, 我们均有

$$r > \left(1 + \frac{\max(|f_1|, |f_2|, \ldots, |f_n|)}{|f_0|}\right)^{-1}.$$

我们再对多项式零点模的大小做一估计, 首先有下面的引理:

引理12.1. 设 $a_{k-1}, \ldots, a_0$ 均非负, 且 $a_{k-1}, \ldots, a_l$ 不全为零, $a_{l-1}, \ldots, a_0$ 不全为零, 则方程

$$P(z) = z^k + a_{k-1}z^{k-1} + \cdots + a_l z^l - a_{l-1}z^{l-1} - \cdots - a_1 z - a_0 = 0$$

有唯一正根 r_0, 且 $P(r) < 0 (\forall 0 < r < r_0)$, $P(r) > 0 (\forall r > r_0)$.

证明. 考虑函数

$$Q(z) = \frac{P(z)}{z^l} = z^{k-l} + a_{k-1}z^{k-l-1} + \cdots + a_{l+1}z + a_l - \frac{a_{l-1}}{z} - \cdots - \frac{a_0}{z^l},$$

易知其在 $(0, +\infty)$ 上单调递增, 且

$$\lim_{z\to 0} Q(z) = -\infty, \quad \lim_{z\to+\infty} Q(z) = +\infty,$$

故 $Q(z)$ 有唯一正实根 r_0, 从而也是 $P(z)$ 的唯一正根, 且由单调性可知 $\{r | r > 0 \wedge P(r) > 0\} = (r_0, +\infty)$. □

定理12.2. 设 $a_{k-1}, \ldots, a_0 \in \mathbb{C}$, r 为多项式方程

$$P(z) = z^k + a_{k-1}z^{k-1} + \cdots + a_1 z + a_0 = 0$$

的任一根, 并设 r_0 是方程

$$Q(z) = z^k - |a_{k-1}|z^{k-1} - \cdots - |a_1|z - |a_0| = 0$$

的唯一正根, 则 $|r| \le r_0$.

再设 r_1 是方程

$$R(z) = z^k + |a_{k-1}|z^{k-1} + \cdots + |a_1|z - |a_0| = 0$$

的唯一正根, 则 $|r| \ge r_1$.

证明. 首先由 $P(r)=0$ 可知

$$\begin{aligned}|r|^k &= |-a_{k-1}r^{k-1}-\cdots-a_1r-a_0|\\ &\le |a_{k-1}||r|^{k-1}+\cdots+|a_1||r|+|a_0|,\end{aligned}$$

即 $Q(|r|)\le 0$, 故 $|r|\le r_0$.

由 r 和 r_1 的定义知道 $1/r$, $1/r_1$ 分别是方程

$$z^k+\frac{a_1}{a_0}z^{k-1}+\cdots+\frac{a_{k-1}}{a_0}z+\frac{1}{a_0}=0$$

和

$$z^k-\frac{|a_1|}{|a_0|}z^{k-1}-\cdots-\frac{|a_{k-1}|}{|a_0|}-\frac{1}{|a_0|}$$

的根, 则根据上面的证明我们有 $1/|r|\le 1/r_1$, 亦即 $|r|\ge r_1$. □

12.2 Jenkins-Traub 算法

12.2.1 算法引入

Traub 最初在一系列论文中提出一种两步骤的迭代算法用以数值求解一元多项式方程, 之后 Jenkins 和 Traub[93] 在此基础上发展出一种三步骤的迭代算法, 下面我们就来对此进行介绍. 该算法收敛比二阶要快.

为了方便起见, 首先我们引入下面一些记号:

定义12.1. 对于我们考虑的多项式 $P(z)=\prod_{j=1}^{k}(z-\alpha_j)^{m_j}\in\mathbb{C}[x]$, 记 $P_j(z)=P(z)/(z-\alpha_j)$, 易知 $P'(z)=\sum_{j=1}^{k}m_jP_j(z)$.

定义12.2. 定义多项式序列 $\{H^{(\lambda)}(z)|\lambda\in\mathbb{N}\}$ 满足

$$\exists c_j^{(\lambda)}\in\mathbb{C}\left(H^{(\lambda)}(z)=\sum_{j=1}^{k}c_j^{(\lambda)}P_j(z)\right).$$

且 $H^{(0)}(z)=P'(z)$. 我们具体构造时可任取某一特定的复数序列 $\{s_\lambda\}$, 使得多项式序列由下面递推关系生成:

$$H^{(\lambda+1)}(z)=\frac{1}{z-s_\lambda}\left[H^{(\lambda)}(z)-\frac{H^{(\lambda)}(s_\lambda)}{P(s_\lambda)}P(z)\right].$$

为了说明用序列 $\{s_\lambda\}$ 来构造多项式序列 $\{H_\lambda\}$ 的合理性, 我们考虑如下关系:

$$\begin{aligned} H^{(\lambda+1)} &= \frac{P(z)}{z-s_\lambda}\left[\sum_{j=1}^{k}\frac{c_j^{(\lambda)}}{z-\alpha_j}-\sum_{j=1}^{k}\frac{c_j^{(\lambda)}}{s_\lambda-\alpha_j}\right] \\ &= \sum_{j=1}^{k}\frac{c_j^{(\lambda)}P(z)}{(z-\alpha_j)(\alpha_j-s_\lambda)} \\ &= \sum_{j=1}^{k}c_j^{(\lambda+1)}P_j(z). \end{aligned}$$

由此可以看出 H_λ 确实如定义所说的那样, 能写成 $P_j(z)$ 的线性组合, 于是这样定义是合理的. 我们顺便得到了系数 $c_j^{(\lambda)}$ 的递推关系:

$$c_j^{(\lambda+1)}=\frac{c_j^{(\lambda)}}{\alpha_j-s_\lambda}=\cdots=\frac{m_j}{\prod_{t=0}^{\lambda}(\alpha_j-s_t)}.$$

Jenkins 和 Traub 给出如下算法, 其中一些具体的细节如 M, L, β 的选取等参见下一节有关注记.

定义12.3. 多项式 $f\in\mathbb{C}[x]$ 的首一化多项式定义为 $\overline{f}=f/\mathrm{lc}(f)$.

算法12.1 (Jenkins-Traub 算法)**.**

步骤 1:No-shift process

$$H^{(0)}(z)=P'(z),$$

$$H^{(\lambda+1)}(z)=\frac{1}{z}\left[H^{(\lambda)}(z)-\frac{H^{(\lambda)}(0)}{P(0)}P(z)\right]\quad(\lambda=0,1,\ldots,M-1).$$

步骤 2:Fixed-shift process

取正数 β 满足 $\beta\le\min_{1\le j\le k}|\alpha_j|$, 随机选取复数 s 满足 $|s|=\beta$, 且 $|s-\alpha_1|\le|s-\alpha_j|(j=2,3,\ldots,k)$, 即 α_1 取为离所选 s 最近的根. 再做如下迭代:

$$H^{(\lambda+1)}(z)=\frac{1}{z-s}\left[H^{(\lambda)}(z)-\frac{H^{(\lambda)}(s)}{P(s)}P(z)\right],\quad(\lambda=M,M+1,\ldots,L-1).$$

步骤 3:Variable-shift process

记 $s_L=s-\frac{P(s)}{H^{(L)}(s)}$, 再做如下迭代:

$$H^{(\lambda+1)}(z)=\frac{1}{z-s_\lambda}\left[H^{(\lambda)}-\frac{H^{(\lambda)}(s_\lambda)}{P(s_\lambda)}P(z)\right],$$

$$s_{\lambda+1}=s_\lambda-\frac{P(s_\lambda)}{\overline{H^{(\lambda+1)}}(s_\lambda)},\quad(\lambda=L,L+1,\cdots).$$

我们得到 $s_\lambda\to\alpha_1$.

定理12.3. 记 $R=\min_{2\le j\le k}|\alpha_1-\alpha_j|$, 若 $|s_L-\alpha_1|<R/2$, $c_1^{(L)}\neq 0$, $D_L=\sum_{j=2}^k|c_j^{(L)}|/|c_1^{(L)}|<1/3$, 则 $s_\lambda\to\alpha_1$.

证明. 若记 $r_j^{(\lambda)}=\dfrac{s_\lambda-\alpha_1}{s_\lambda-\alpha_j}$, $d_j^{(\lambda)}=\dfrac{c_j^{(\lambda)}}{c_1^{(\lambda)}}$, $T_\lambda=\left|\dfrac{s_{\lambda+1}-\alpha_1}{s_\lambda-\alpha_1}\right|$, 则我们只需要证明存在 τ 使得 $\forall\lambda\ge L(T_\lambda\le\tau<1)$ 即可证明收敛性.

而

$$\begin{aligned}\frac{s_{\lambda+1}-\alpha_1}{s_\lambda-\alpha_1}=&\frac{s_\lambda-\frac{P(s_\lambda)}{\overline{H^{(\lambda+1)}}(s_\lambda)}-\alpha_1}{s_\lambda-\alpha_1}\\&=1-\frac{P(s_\lambda)}{\overline{H^{(\lambda+1)}}(s_\lambda)(s_\lambda-\alpha_1)}\\&=1-\frac{P(s_\lambda)\sum_{1\le j\le k}c_j^{(\lambda)}/(\alpha_j-s_\lambda)}{P(s_\lambda)\sum_{1\le j\le k}\frac{c_j^{(\lambda)}}{(\alpha_j-s_\lambda)(s_\lambda-\alpha_j)}(s_\lambda-\alpha_1)}\\&=1-\frac{\sum_{1\le j\le k}c_j^{(\lambda)}(s_\lambda-\alpha_1)/(s_\lambda-\alpha_j)}{\sum_{1\le j\le k}c_j^{(\lambda)}(s_\lambda-\alpha_1)^2/(s_\lambda-\alpha_j)^2}\\&=1-\frac{1+\sum_{2\le j\le k}d_j^{(\lambda)}r_j^{(\lambda)}}{1+\sum_{2\le j\le k}d_j^{(\lambda)}[r_j^{(\lambda)}]^2}\\&=\frac{\sum_{2\le j\le k}[r_j^{(\lambda)}]^2d_j^{(\lambda)}-\sum_{2\le j\le k}r_j^{(\lambda)}d_j^{(\lambda)}}{1+\sum_{2\le j\le k}[r_j^{(\lambda)}]^2d_j^{(\lambda)}},\end{aligned}$$

由于 $|r_j^{(L)}|<1$, 则

$$T_L\le\frac{\sum_{2\le j\le k}|d_j^{(L)}|+\sum_{2\le j\le k}|d_j^{(L)}|}{1-\sum_{2\le j\le k}|d_j^{(L)}|}=\frac{2D_L}{1-D_L}<1,$$

令 $\tau=2D_L/(1-D_L)$, 即得 $T_L\le\tau<1$.

假设 $T_L,T_{L+1},\ldots,T_{\lambda-1}\le\tau<1$, 则对 $t=L,L+1,\ldots,\lambda$, 有

$$|s_t-\alpha_1|\le|s_L-\alpha_1|<R/2,$$

$$|s_t-\alpha_j|\geq|\alpha_1-\alpha_j|-|s_t-\alpha_1|>R/2,$$

即仍有 $|r_j^{(t)}|<1(t=L,L+1,\ldots,\lambda)$, 又由于

$$d_j^{(\lambda)}=\frac{c_j^{(\lambda)}}{c_1^{(\lambda)}}=\frac{c_j^{(\lambda-1)}}{\alpha_j-s_{\lambda-1}}\frac{\alpha_1-s_{\lambda-1}}{c_1^{(\lambda-1)}}=r_j^{(\lambda-1)}d_j^{(\lambda-1)},$$

则 $\sum_{2\leq j\leq k}|d_j^{(\lambda)}|\leq D_L$, 于是 $T_\lambda\leq\tau<1$, 从而我们归纳证明了 $T_\lambda\leq\tau<1(\forall\lambda\geq L)$.

为了说明迭代的合理性, 我们仍要证明 $H^{(\lambda)}(s_\lambda)\neq0$, 因为

$$\begin{aligned}\overline{H^{(\lambda)}}(s_\lambda)&=\frac{\sum_{1\leq j\leq k}c_j^{(\lambda)}/(\alpha_j-s_\lambda)\cdot P_j(s_\lambda)}{\sum_{1\leq j\leq k}c_j^{(\lambda)}/(\alpha_j-s_\lambda)}\\&=P_1(s_\lambda)\left[\frac{1+\sum_{2\leq j\leq k}d_j^{(\lambda)}[r_j^{(\lambda)}]^2}{1+\sum_{2\leq j\leq k}d_j^{(\lambda)}r_j^{(\lambda)}}\right],\end{aligned}$$

我们假定 $P(s_\lambda)\neq0$, 且又由 $|\sum_{2\leq j\leq k}d_j^{(\lambda)}[r_j^{(\lambda)}]^2|<1/3$ 知 $|\overline{H^{(\lambda+1)}}(s_\lambda)|>0$. □

有了上面的定理, 下面证明收敛性:

定理12.4. *令 s 为满足算法 12.1 步骤 2 中条件的复数, 则当迭代步数 L 足够大时, 步骤 3 中迭代是收敛的.*

证明. 很容易知道:

$$H^{(L)}(z)=\sum_{1\leq j\leq k}m_j\alpha_j^{-M}(\alpha_j-s)^{-(L-M)}P_j(z)=\sum_{1\leq j\leq k}c_j^{(L)}p_j(z),$$

于是

$$\sum_{2\leq j\leq k}d_j^{(L)}=\sum_{2\leq j\leq k}\frac{m_j}{m_1}\left(\frac{\alpha_1}{\alpha_j}\right)^M\left(\frac{\alpha_1-s}{\alpha_j-s}\right)^{L-M},$$

固定 M 后, 取 L 充分大, 可使上式足够小, 故我们可以取到 L 使得 $D_L<1/3$.

我们再取 L 足够大使 $2D_L/(1-D_L)$ 足够小使 $|s_L-\alpha_1|=|s-\alpha_1|\frac{2D_L}{1-D_L}<R/2$, 则前述定理条件均满足, 由是, $s_\lambda\to\alpha_1$. □

12.2.2 收敛速度和一些细节说明

下面给出对收敛速度的估计,

定理12.5. *设 $D_L<1/3$, $c_1^{(L)}\neq0$, $|s_L-\alpha_1|<R/2$, 则*

$$C(\lambda)=\frac{|s_{L+\lambda+1}-\alpha_1|}{|s_{L+\lambda}-\alpha_1|^2}\leq\frac{2}{R}\tau^{\lambda(\lambda-1)/2}.$$

证明. 首先

$$\frac{s_{L+\lambda+1}-\alpha-1}{s_{L+\lambda}-\alpha-1}=\frac{\sum_{2\le j\le k}\frac{r_j^{(L+\lambda)}d_j^{(L+\lambda)}}{s_{L+\lambda}-\alpha_j}-\sum_{2\le j\le k}\frac{d_j^{(L+\lambda)}}{s_{L+\lambda}-\alpha_j}}{1+\sum_{2\le j\le k}[r_j^{(L+\lambda)}]^2d_j^{(L+\lambda)}},$$

由于 $|s_L-\alpha_1|<R/2$, 则 $|s_{L+\lambda}-\alpha_1|<\tau^\lambda R/2$, 又 $|s_{L+\lambda}-\alpha_j|>R/2$, 则 $|r_j^{(L+\lambda)}|<\tau^\lambda$. 于是

$$|d_j^{(L+\lambda)}|=|r_j^{(L+\lambda-1)}||d_j^{(L+\lambda-1)}|=\cdots\le\tau^{\lambda-1}\tau^{\lambda-2}\cdots\tau|d_j^{(L)}|=\tau^{(\lambda-1)\lambda/2}|d_j^{(L)}|,$$

则 $\sum_{2\le j\le k}|d_j^{(L+\lambda)}|\le\tau^{\lambda(\lambda-1)/2}D_L\le\frac{1}{3}\tau^{\lambda(\lambda-1)/2}$. 且由 $\frac{1}{|s_{L+\lambda}-\alpha_j|}\le\frac{2}{R}$, 代入前面表达式可得

$$C(\lambda)\le\frac{2}{R}\tau^{\lambda(\lambda-1)/2}.$$

证毕. □

由此可以看到, Jenkins-Traub 算法是至少二阶收敛的, 它比普通的 Newton 迭代法要快.

推论12.2. *当上面定理条件满足时, 对于 $\lambda\ge1$ 有 $|s_{L+\lambda}-\alpha_1|\le\frac{1}{2}R\tau^\eta$, 其中 $\eta=\frac{1}{2}[3\cdot2^\lambda-(\lambda^2+\lambda+2)]$.*

对于算法细节的说明:

注159. 步骤 1 中 M 的选取是非必需的, 此处只是要强调模较小的零点. 计算的经验表明一般取 $M=5$ 较合适.

注160. 关于在步骤 2 中对于方程 $P(z)=z^k+a_{k-1}z^{k-1}+\cdots+a_1z+a_0=0$ 零点最小模的估计, 根据定理 12.2 , 我们取 β 为方程 $z^k+|a_{k-1}|z^{k-1}+\cdots+|a_1|z-|a_0|=0$ 的唯一正根, 该根可由 Newton 迭代求出.

注161. 关于步骤 2 的终止, 即 L 的选取, 前面给出的应当来说是一个充分条件, 而且在算法实现时我们并不知道所有根的分布情况, 因此在实践中取下面的收敛性判别条件: 令 $t_\lambda=s-\frac{P(s)}{H^{(\lambda)}(s)}$, 若在一定迭代步数(例如 20 步)内能够满足下面条件:

$$|t_{\lambda+1}-t_\lambda|\le|t_\lambda|/2,\quad|t_{\lambda+2}-t_{\lambda+1}|\le|t_{\lambda+1}|/2,$$

则终止.

注162. 步骤 3 终止的条件根据计算精度要求决定.

步骤 3 中的迭代事实上是 Newton 迭代, 我们有下面的

定理12.6. 设 $w^{(\lambda)}(z)=\frac{P(z)}{H^{(\lambda)}(z)}$, 则每一步迭代过程相当于

$$s_{\lambda+1}=s_\lambda-\frac{P(s_\lambda)}{H^{(\lambda+1)}(s_\lambda)}=s_\lambda-\frac{w^\lambda(s_\lambda)}{(w^\lambda)'(s_\lambda)}.$$

证明. 定义 $v^{(\lambda)}(z)=\frac{H^{(\lambda)}(z)}{P(z)}=1/w^{(\lambda)}(z)$, 则由 $H^{(\lambda)}(z)$ 的递推生成式可知:$v^{(\lambda+1)}=(v^{(\lambda)})'$, $\mathrm{lc}(H^{(\lambda+1)}(z))=-v^{(\lambda)}(s_\lambda)$.

于是

$$\begin{aligned}s_{\lambda+1}&=s_\lambda-\frac{P(s_\lambda)\mathrm{lc}(H^{(\lambda+1)}(z))}{H^{(\lambda+1)}(s_\lambda)}=s_\lambda+\frac{v^{(\lambda)}(s_\lambda)}{v^{(\lambda+1)}(s_\lambda)}=s_\lambda+\frac{v^{(\lambda)}(s_\lambda)}{(v^{(\lambda)}(s_\lambda))'}\\&=s_\lambda-\frac{w^{(\lambda)}(s_\lambda)}{(w^{(\lambda)}(s_\lambda))'}.\end{aligned}$$

证毕. □

对实一元多项式, Jenkins 和 Traub[92] 扩展了已有的迭代算法, 发展了一套二次因子的迭代算法, 仅仅利用实数运算计算多项式的所有根(包括共轭复根), 当将两种 Jenkins-Traub 算法应用于同一实一元多项式上时, 新算法能将效率提高达四倍.

12.3 Laguerre 算法

一元多项式数值求根的迭代算法有很多, 利如 Bairstow 算法, Graeffe 算法, Müller 算法, Laguerre 算法等, [122] 对各个算法均有介绍, 有兴趣的读者可以参考该文以及文后所引的参考文献.

下面我们简单介绍 Laguerre 算法, 它也是一种比 Newton 迭代快的算法([122]2.9 节), 用到了多项式的二阶导数. 考虑复多项式 $P(z)$, 其有 n 个根 $r_1,r_2,\ldots,r_n$, 定义如下一些函数:

$$S_1(z)=\frac{P'(z)}{P(z)}=\sum_{i=1}^{n}\frac{1}{z-r_i},$$

$$S_2(z)=-S_1'(z)=\sum_{i=1}^{n}\frac{1}{(z-r_i)^2},$$

对于某个固定的 j, 记

$$\alpha(z)=\frac{1}{z-r_j},\quad \beta(z)=\frac{1}{n-1}\sum_{1\le i\le n,i\ne j}\frac{1}{z-r_i},\quad \delta_i(z)=\frac{1}{z-r_i}-\beta(z),(i\ne j),$$

于是 $S_1=\alpha+(n-1)\beta$, 若定义 $\delta^2=\sum_{i\neq j}\delta_i^2$, 则还有

$$\begin{aligned}S_2&=\alpha^2+\sum_{i\neq j}(\beta+\delta_i)^2=\alpha^2+(n-1)\beta^2+2\beta\sum_{i\neq j}\delta_i+\sum_{i\neq j}\delta_i^2\\&=\alpha^2+(n-1)\beta^2+\delta^2.\end{aligned}$$

若消去 β, 可得到关于 α 的二次方程:

$$n\alpha^2-2S_1\alpha+S_1^2-(n-1)(S_2-\delta^2)=0$$

解之得

$$\alpha=\frac{S_1\pm\sqrt{(n-1)(nS_2-S_1^2-n\delta^2)}}{n}.$$

由 α 的定义我们可以得到

$$r_j=z-\frac{n}{S_1\pm\sqrt{(n-1)(nS_2-S_1^2-n\delta^2)}},$$

但若我们令 $\delta^2=0$, 注意到当 z 接近零点 r_j 时, 在 S_1, S_2 表达式中各项只有含 r_j 的一项占主导地位, 是奇异部分, 因此迭代过程中这样的假设是合理的. 于是我们得到逼近 r_j 的序列 $\{z_j^{(k)}\}$ 的迭代式, 即 Laguerre 迭代公式:

$$z_j^{(k+1)}=z_j^{(k)}-\frac{n}{S_1\pm\sqrt{(n-1)(nS_2-S_1^2)}}.$$

该迭代算法对于单根是三阶收敛, 对于多重根是一阶线性收敛. 下面的改进算法给出单根时的四阶收敛:

$$z_j^{(k+1)}=z_j^{(k)}-\frac{n}{S_1\pm\sqrt{(n-1)(nS_2-S_1^2-n\delta_j^2)}},$$

其中

$$\delta_j=\sum_{i\neq j}\left[\frac{1}{z_j^{(k)}-z_i^{(k)}}-\beta_j\right]^2,\quad \beta_j=\frac{1}{n-1}\sum_{i\neq j}\frac{1}{z_j^{(k)}-z_i^{(k)}}.$$

12.4 代数模方程求解

这一小节我们讨论有限域中的代数方程求解. 对于一次模方程, 我们可以很简单地直接解出, 下面我们先介绍一下有限域中的开平方算法, 这些均与数论中的二次剩余理论有联系.

12.4.1 $\mathbb{F}_p$ 中的开平方算法

先看一些比较特殊的情况. 我们考虑问题 $x^2 \equiv a \pmod p$, 其中 a 是模 p 的二次剩余, 其 Jacobi 符号 $\left(\frac{a}{p}\right) = 1$. 此时必有 $a^{(p-1)/2} \equiv 1 \pmod p$.

若素数 $p \equiv 3 \pmod 4$, 则令 $x \equiv a^{(p+1)/4} \pmod p$ 即可.

若素数 $p \equiv 5 \pmod 8$, 此时 $a^{(p-1)/4} \equiv \pm 1 \pmod p$, 若取正号, 则命 $x \equiv a^{(p+3)/8} \pmod p$, 否则由 $p \equiv 5 \pmod 8$ 有 $2^{(p-1)/2} \equiv -1 \pmod p$, 于是命 $x \equiv 2a(4a)^{(p-5)/8} \pmod p$.

对于一般情况, 我们可以尝试有限域上的因子分解算法. Schoof 提出了一种多项式时间的非概率性方法, 因为它要利用椭圆曲线, 过程过于复杂, 所以我们介绍另一种概率性的 Tonelli & Shanks 算法 [56].

首先我们可以将 $p-1$ 中的 2 的幂次分离出来, 即存在 e, q, q 是奇数, e 是自然数使得 $p - 1 = 2^e q$. 因为乘法群 $\mathbb{F}_p^*$ 与加群 $\mathbb{Z}/(p-1)\mathbb{Z}$ 同构, 则其 Sylow 2-子群 G 是一个 2^e 阶的循环群, 设 z 是 G 的生成元, 则 G 中平方数的阶整除 2^{e-1} 且是 z 的偶数次幂.

当 a 是模 p 中的二次剩余时, 我们有 $a^{(p-1)/2} = (a^q)^{(2^{e-1})} \equiv 1 \pmod p$, 于是 $b = a^q \bmod p$ 是 G 中的平方数, 故存在偶数 $k (0 \le k < 2^e)$ 使得 $a^q z^k = 1$. 此时若令

$$x = a^{(q+1)/2} z^{k/2},$$

则有 $x^2 = a^{q+1} z^k \equiv a \pmod p$.

下面给出求平方根的算法:

算法12.2 (Shanks 算法).

输入:奇素数 p, a, 以及 e, q 使得 $p - 1 = 2^e q$, q 是奇数,

输出:x 满足 $x^2 \equiv a \pmod p$, 或者不存在.

1. 随机选取 n 使得 $\left(\frac{n}{p}\right) = -1$, 令 $z = n^q \bmod p$,
2. 令 $y = z$, $r = e$, $x = a^{(q-1)/2} \bmod p$, $b = ax^2 \bmod p$, $x = ax \bmod p$,
3. 若 $b \equiv 1 \pmod p$, 则输出 x 并终止, 否则找到最小的 $m \ge 1$ 使得 $b^{2^m} \equiv 1 \pmod p$, 若 $m = r$, 则输出 a 非二次剩余并终止,
4. 令 $t = y^{2^{r-m-1}} \bmod p$, $y = t^2 \bmod p$, $r = m \bmod p$, $x = xt \bmod p$, $b = by \bmod p$, 并转 3 步.

注163. 算法第 1 步是用随机算法求 G 的生成元 z. 可以看出 z 是生成元当且仅当 n 非模 p 二次剩余. (等价于 $z^{2^{e-1}} \equiv -1 \pmod{p}$)

注164. 显式地找 k 比较困难, 因此 Shanks 提出了上面的算法. 注意到在算法开始时, 有下面的等式成立:

$$ab = x^2, \quad y^{2^{r-1}} = -1, \quad b^{2^{r-1}} = 1.$$

记 G_r 是群 G 中阶整除 2^r 的元素组成的子群, 则 y 是 G_r 的生成元且 $b \in G_{r-1}$, 因此 b 是群 G_r 中的二次剩余.

显然每次循环之后, r 都会严格地减小. 设某次循环前各量用下标 0 表示, 循环后各量用下标 1 表示, 则 $b_1 = b_0 y_0^{2^{r-m}}$, $x_1 = x_0 y_0^{2^{r-m-1}}$, $y_1 = y_0^{2^{r-m}}$, 于是

$$ab_1 = ab_0 y_0^{2^{r-m}} = x_0^2 y_0^{2^{r-m}} = x_1^2,$$

$$y_1^{2^{m-1}} = y_0^{2^{r-1}} = -1,$$

$$b_1^{2^{m-1}} = b_0^{2^{m-1}} y_0^{2^{r-1}} = (-1)(-1) = 1,$$

从而由归纳可知每次循环后上面三个等式均成立. 当 $r \leq 1$ 时, 我们有 $b = 1$, 于是算法是可终止的.

例12.1. 求 x 使 $x^2 \equiv 10 \pmod{13}$.

解: $13 - 1 = 2^2 \times 3$, 故 $e = 2, q = 3$. 取 $n = 2$, 则 $\left(\frac{2}{13}\right) = -1$, $z = 2^3 \bmod 13 = 8$.

首先令 $y = 8$, $r = 2$, $x = 10^{(3-1)/2} \bmod 13 = 10$, $b = 10 \times 10^2 \bmod 13 = 12$, $x = 10 \times 10 \bmod 13 = 9$.

因为 $b \neq 1$, 且使 $b^{2^m} = 1$ 的最小的 m 为 $m = 1$, 故 $t = 8^{2^{2-1-1}} \bmod 13 = 8$, $y = 8^2 \bmod 13 = 12$, $r = 1$, $x = 9 \times 8 \bmod 13 = 7$, $b = 12 \times 12 \bmod 13 = 1$. 故此时输出 $x = 7$. ◇

12.4.2 模 p 代数方程求解

实际上在域中解多项式方程时, 可以看作是求多项式的因子分解问题. 而在有限域上, 因子分解算法本身是简单的(见有限域上因子分解有关章节, 事实上当时提出了一个 $\mathbb{F}_p$ 上代数方程求根算法), 因此我们期望用因子分解的算法来求根. 事实上我们将看到下面给出的求根算法 [56] 就是有限域因子分解算法. 我们将在给出算法后再对算法中的每一步进行分析.

算法12.3 (模 p 代数方程求根算法).

输入:素数 $p \geq 3$, $f \in \mathbb{F}_p[x]$,

输出:$\mathbb{F}_p$ 中 f 的根.

1. 求 $g = \gcd(x^p - x, f)$, 若 $g(0) = 0$, 输出 0 并令 $g = g/x$,
2. 若 $\deg g = 0$, 则结束, 若 $\deg g = 1$, 即 $g = g_1 x + g_0$, 则输出 $-g_0/g_1$ 并终止, 若 $\deg g = 2$, $g = g_2 x^2 + g_1 x + g_0$, 则令 $d = g_1^2 - 4g_0 g_2$, 计算 $e = \sqrt{d}$, 输出 $\dfrac{-g_1 \pm e}{2g_2}$ 并终止,
3. 随机取 $a \in \mathbb{F}_p$, $h(x) = \gcd((x+a)^{(p-1)/2} - 1, g)$, 若 $\deg h = 0$ 或 $\deg h = \deg g$, 则重新执行此步,
4. 递归调用本算法输出 b 和 a/b 的根, 注意调用时不必再执行第 1 步.

注165. 第 1 步实际上是将 f 中一次不可约因子的乘积提取出来, 只考虑在 $\mathbb{F}_p$ 中的根, 并且将 0 根做了预处理, 从而从 g 中排除掉.

第 2 步是进行一些平凡的处理, 即对于一次和二次的 g 直接由代数运算或求根公式求得其根.

第 3 步实际上是同次因子分解算法. 只不过这里随机取的多项式是 $x + a$, 因为一次因子的幂次计算起来要容易一些 [56].

注166. 该算法与同次因子分解算法不同之处再于对于二次多项式有一个直接处理过程, 而不必再通过概率算法分解为两个一次因子.

12.5 实一元多项式实根隔离算法

12.5.1 Sturm 序列

Sturm 序列可用来在实轴上隔离实一元多项式的根, 这一节我们先介绍这方面的理论. 首先定义(见 [14]170 页):

定义12.4 (广义 Sturm 序列). 我们考虑闭区间 $[a, b]$ 上一无平方因子实多项式 p, 称 $p_0(= p), \ldots, p_k$ 为广义 Sturm 序列, 如果它们满足

1. $p(a)p(b) \neq 0$,
1. p_k 在 $[a, b]$ 上不变号,

2. 若 $p_i(\xi)=0(1\le i\le k-1,\xi\in[a,b])$, 则 $p_{i-1}(\xi)p_{i+1}(\xi)<0$

3. 对于 $c\in[a,b]$, 若 $p(c)=0$, 则 $p(x)p_1(x)$ 在 c 的邻域内与 $x-c$ 符号相同.

此时对于 $y\in\mathbb{R}$, 定义 $V(y)$ 为序列 $p_0(y),p_1(y),\ldots,p_k(y)$ 的变号次数, 即

$$V(y)=\#\{i|p_i(y)p_{i+1}(y)<0\}.$$

对于 $y=\pm\infty$ 可在极限意义下类似定义.

广义 Sturm 序列有如下性质:

定理12.7. *对于 $[a,b]$ 上的广义 Sturm 序列, $V(a)-V(b)$ 为 p 在 $[a,b]$ 上实根的个数.*

证明. 设 $x_1,x_2\in[a,b]$, $x_1<x_2$, 若 $[x_1,x_2]$ 中无 $p_i(0\le i\le k)$ 的根, 则 $V(x_1)=V(x_2)$, 此时函数 $V(x)$ 不发生变化.

1. 设 $a<c<b$ 且 $p(c)=0$, 则由定义中第 4 条知存在 $\varepsilon>0$ 使得 $x\in(c-\varepsilon,c)$ 时, $p(x)p_1(x)<0$, 此区间内变一次号, 当 $x\in(c,c+\varepsilon)$ 时 $p(x)p_1(x)>0$, 此区间内不变号, 此时若将 $V(x)$ 限定在从 $p(x)$ 到 $p_1(x)$ 的变号, 则有 $V(x)$ 会减小 1.

2. 设 $a<c<b$ 且对某个 $i(1\le i\le k-1)$, $p_i(c)=0$, 则由定义中第 3 条知 $x\in(c-\varepsilon,c+\varepsilon)$ 时, $p_{i-1}(x)p_{i+1}(x)<0$, 此时若将 $V(x)$ 限定在从 $p_{i-1}(x)$ 到 $p_{i+1}(x)$ 的变号上时, 函数 $V(x)$ 在小区间上不变.

证毕. □

需要郑重说明的一点是, 从上面定理的证明过程来看, 实际上我们不仅要求 p 在 a,b 两点上不为零, 还要求诸 p_i 在此两点也不为零. 因为根据我们对于变号的定义, 序列

$$-1,\pm\varepsilon,1$$

的变号数为 1, 对于连续函数情形我们可以对 x 进行微小移动使得序列成为

$$-1+\varepsilon_1,0,1+\varepsilon_2,$$

则变号数变为 0. 于是对于序列中各个多项式 p_i, 需要满足 $p_i(a)p_i(b)\neq 0$.

如何找到一个这样的广义 Sturm 序列呢? 下面的定义给出了一个具体的实例.

定义12.5 (Sturm 序列). 对于无平方因子多项式 $p \in \mathbb{R}[x]$, 其 Sturm 序列是指多项式序列 $p_0(x) = p, p_1(x), \ldots, p_k(x) \in \mathbb{R}[x]$, 其中

$$p_1 = p', p_i = -p_{i-2} \bmod p_{i-1} (2 \leq i \leq k),$$

直至 $p_k(x)$ 为一常数多项式.

定理12.8. *Sturm 序列是广义 Sturm 序列.*

证明. 第一个条件 $p(a)p(b) \neq 0$ 我们总可以取到, 只需要取一组满足条件的 a, b 即可.

第二个条件由 $p_k \in \mathbb{R}$ 也可得到.

第三个条件:对于 $p_i(\xi) = 0 (\xi \in [a, b])$, 首先 $p_{i-1}(\xi)p_{i+1}(\xi) \neq 0$. 倘若其中有一个是零, 不妨设 $p_{i-1}(\xi) = 0$, 则 $(x - \xi) | \gcd(p_{i-1}, p_i) = \gcd(p, p') \Rightarrow p$ 有重因子, 矛盾. 再由 $p_{i+1} = -p_{i-1} \bmod p_i$ 知 $p_{i-1}(\xi)p_{i+1}(\xi) < 0$.

第四个条件:对于 $p(c) = 0 (c \in [a, b])$, 由于其无平方, $p_1(c) = p'(c) \neq 0$, 由导数的定义知

$$p_1(c) = \lim_{x \to c} \frac{p(x) - p(c)}{x - c} = \lim_{x \to c} \frac{p(x)}{x - c},$$

则在 c 的某个去心邻域内有 $p(x)/(x - c)$ 与 $p_1(x)$ 同号. □

由上面的定理可以得到如下推论:

定理12.9 (Sturm 定理). *p 是无平方多项式, $V(x)$ 是 p 的 Sturm 序列在 x 点的变号数, 则 p 在区间 $[a, b]$ 上实根的个数为 $V(a) - V(b)$.*

这里变号数 $V(x)$ 定义为:

$$V(x) = \#\{i | p_i(x)p_{i+1}(x) < 0\} + \#\{i | p_i(x) = 0\}.$$

我们对这里变号数 $V(x)$ 的重新定义做一些说明. 前文我们已经说过, 对于各个 p_i 也需要它们在 a, b 两点不为零. 事实上, 假若 p_i 在 a 点值为零, 则在 a 的足够小的邻域内是可以得到正确变号数的. 其实若根据广义 Sturm 序列满足的第 3 个条件可知, 此时必有 $p_{i-1}(a)p_{i+1}(a) < 0$, 此处 $p_{i-1}(a), p_i(a), p_{i+1}(a)$ 应取为 1. 显然我们可以得到重新定义的 $V(x)$ 的表达式.

12.5.2 由 Sturm 序列给出的实根隔离算法

区间隔离算法本质上是一种分治法的思想. 我们利用 Sturm 序列不断将根的隔离, 最终将每个根都隔离开(见 [13]).

算法12.4 (实根隔离算法).

输入:无平方因子多项式 p, 区间 $[x_1, x_2]$, 且 $p(x_1)p(x_2) \neq 0$,

输出:$[x_1, x_2]$ 上所有根的隔离区间的集合 $S = \{[a_1, b_1], [a_2, b_2], \ldots, [a_k, b_k]\}$, 其中 $[a_i, b_i](1 \leq i \leq k)$ 中含有且仅含有一个 p 的实根.

1. $T = \{[x_1, x_2]\}$,
2. 任取区间 $[a, b] \in T$, 令 $T = T \setminus \{[a, b]\}$, 若 $V(a) - V(b) = 1$, 则 $S = S \bigcup \{[a, b]\}$, 转 6 步, 若 $V(a) - V(b) = 0$ 则直接转第 6 步,
3. 此时必有 $V(a) - V(b) > 1$, 令 $c = (a + b)/2$, $T = T \setminus \{[a, b]\}$,
4. 若 $p(c) \neq 0$, 考虑 $V(a) - V(c)$, $V(c) - V(b)$, 若 $V(a) - V(c) = 1$ 则 $S = S \bigcup \{[a, c]\}$, 否则 $T = T \bigcup \{[a, c]\}$, 对于 $V(c) - V(b)$ 和 $[c, b]$ 同样操作, 转 6 步,
5. 否则有 $p(c) = 0$, $S = S \bigcup \{[c, c]\}$, 作代换 $y = x - c$, 并令 $p_1(y) = p(y + c)/y$, 则 $p_1(0) \neq 0$, 求其根的绝对值的下界 M, 则对于 p 有 $V(c-M) = V(c+M)+1$. 若 $V(a)-V(c-M) = 1$ 则 $S = S \bigcup \{[a, c-M]\}$, 否则 $T = T \bigcup \{[a, c-M]\}$. 同样利用 $V(c + M) - V(b)$ 来决定将 $[c + M, b]$ 放在 S 或 T 中,
6. 若 $T = \varnothing$ 则输出 S, 否则转 2 步.

接下来我们可以用二分法缩小区间, 或用 Newton 迭代法求根的数值解.

12.6 分圆多项式

本章开头我们已经提到过, 对于“原子”级的多项式, 即不可约并且不可进行复合分解的多项式, 可以得到根式解或者符号解的只有不高于四次的多项式以及的分圆多项式. 对于前者有求根公式, 可以参考相关代数书或者数学手册. 这里我们着重介绍分圆多项式的检测.

12.6.1 分圆多项式的定义及生成

定义12.6. 定义多项式

$$\Phi_n = \prod_{\substack{1 \leq k < n \\ \gcd(k,n)=1}} (x - e^{2\pi i k/n})$$

为 n 阶分圆多项式(nth cyclotomic polynomial).

定理12.10. *分圆多项式都是整系数不可约多项式(见 [6]224–227 页).*

很容易看出, Φ_n 的次数为 $\phi(n)$, ϕ 为 Euler 函数. 分圆多项式与 n 次单位根有很大的联系, 我们有下面的引理.

引理12.2. $x^n-1=\prod_{d|n}\Phi_d$.

证明. 不妨设 ω 为一 n 次单位根, 其阶为 d, 显然 $d|n$, 并且 $\Phi_d(\omega)=1$. 由此可以知道欲证等式的两端有相同的根. 再由两端均是无平方因子多项式且首一, 故等式成立. □

引理的等式可以写为 $\ln(x^n-1)=\sum_{d|n}\ln\Phi_d$, 由 Mobius 反演变换可得 $\ln\Phi_n=\sum_{d|n}\mu(n/d)\ln(x^d-1)$, 即

$$\Phi_n=\prod_{d|n}(x^d-1)^{\mu(n/d)}.$$

推论12.3. $\Phi_n=\prod_{d|n}(x^d-1)^{\mu(n/d)}$.

引理12.3. *设 n,k 是正整数, 我们有:*

1. *若 n 是素数, 则 $\Phi_n=x^{n-1}+x^{n-2}+\cdots+x+1$,*
2. *若 n 是奇数, 则 $\Phi_{2n}=\Phi_n(-x)$,*
3. *若 $\gcd(k,n)=1$, 则 $\Phi_{kn}\Phi_n=\Phi_n(x^k)$,*
4. *若 k 的素因子均整除 n, 则 $\Phi_{kn}=\Phi_n(x^k)$.*

证明. 逐条证明如下:

(1)当 n 是素数时, 由 $\phi(n)=n-1$ 即得.

(2)可以验证 ω 的阶是 n 当且仅当 $-\omega$ 的阶是 $2n$.

(3)当 $\gcd(k,n)=1$ 时有 $\phi(kn)=\phi(k)\phi(n)=(k-1)\phi(n)$(假设 k 是一素数). 若 ω 的阶是 kn, 则 ω^k 的阶是 n. 同样地若 ω 的阶是 n, 则 ω^k 的阶也是 n, 因此由二者均无平方因子, 次数相同且首一可知等号成立.

(4)由条件可知 $\phi(kn)=k\phi(n)$. 若 ω 的阶是 kn, 可得 ω^k 的阶是 n. 同样由二者均无平方因子, 次数相同且首一可知等号成立. □

引理表明各分圆多项式实际上都是整系数的, 并且我们可以构造如下分圆多项式生成算法[174]:

算法12.5 (分圆多项式的生成算法).

输入:正整数 n 和它的互不相同的素因子 $p_1,\ldots,p_r$,

输出:n 阶分圆多项式 Φ_n.

1. $f_0=x-1$,
2. 对 i 从 1 循环到 r, 做 $f_i=\dfrac{f_{i-1}(x^{p_i})}{f_{i-1}}$,
3. 输出 $f_r(x^{n/(p_1p_2\cdots p_r)})$.

12.6.2　分圆多项式的 Graeffe 检测方法

要精确求解多项式方程, 我们就要能有效地进行分圆多项式检测. [37] 提出了两种有效的检测方法, 并给出了有位移的分圆多项式(Shifted cyclotomic polynomial)的检测方法. 下面介绍其中一个算法:Graeffe 方法. 下一节将介绍另一算法:Euler 反函数(Inverse ϕ)方法. 我们在以下三个小节中分别介绍.

算法12.6 (Graeffe 过程).

输入:多项式 f,

输出:多项式 $f_1=\mathrm{graeffe}(f)$, 其根的集合为 f 的根的平方的集合.

1. 将 $f(x)$ 表达为奇偶两部分和的形式, 即 $f(x)=g(x^2)+xh(x^2)$, 其中 $g(x^2)$ 和 $xh(x^2)$ 分别是 $f(x)$ 的偶和奇函数部分,
2. 令 $f_1(x)=g(x)^2-xh(x)^2$,
3. 乘以适当常数使 $\mathrm{lc}(f_1)$ 为正数并输出.

算法有效性. 设 $f(x)=0$, 则 $f_1(x^2)=g(x^2)^2-x^2h(x^2)^2=(g(x^2)-xh(x^2))(g(x^2)+xh(x^2))=0$.

设 $f_1(x^2)=0$, 则我们有 $g(x^2)-xh(x^2)=0$ 或 $g(x^2)+xh(x^2)=0$, 无论哪种情形, 均有 $f(x)=0$ 或 $f(-x)=0$.

由证明过程还可以看出, 这两者之间是一一对应的. □

算法12.7 (Graeffe 检测方法).

输入:不可约多项式 $f \in \mathbb{Z}[x]$,

输出:f 是否是分圆多项式.

设 $f_1 = \text{graeffe}(f)$, 则:

1. 若 $f_1(x) = f(x)$, 则 f 是分圆多项式.
2. 若 $f_1(x) = f(-x)$, 且 $f(-x)$ 是分圆多项式, 则 f 是分圆多项式.
3. 若 $f_1 = f_2^2$, 其中 f_2 是分圆多项式, 则 f 是分圆多项式.
4. 对于其它情况, f 均不是分圆多项式.

算法有效性. (1)任取 f 的一个根 α, 由 $f_1 = f$ 可知 $\alpha^2, \alpha^4, \ldots, \alpha^{2^k}, \cdots$ 均是 f 的根, 故必存在 $k \geq 1$ 使得 $\alpha^k = 1$. 再由 f 的不可约性可知 f 是以 α 为一本原单位根生成的分圆多项式.

(2)若 n 是奇数, 则 $(-x)^n - 1 = -(x^n + 1) | (x^{2n} - 1)$, 否则 $(-x)^n - 1 = x^n - 1 | (x^{2n} - 1)$, 无论哪种情况均可由 $f(-x)$ 是分圆多项式得到 $f(x)$ 是分圆多项式.

(3)此时 f 的根是一个分圆多项式根的平方根, 且 f 不可约, 因此 f 也是分圆多项式.

(4)反过来我们设 f 是一个分圆多项式, 设 $f | x^n - 1$, 若 n 是奇数, 则将 2 乘到 n 的简化剩余系上仍然是一个简化剩余系, 即 f 的根的平方均列出了 f 的根, 因此 $f_1 = f$. 若 $n = 2q$, q 是奇数, 则 f_1 的根是 q 次的本原单位根, 故其相反数是 $n = 2q$ 次本原单位根. 若 $4 | n$, 由于 f_1 的根均是 $n/2$ 次本原单位根, 但是 $\phi(n) = 2\phi(n/4) = 2\phi(n/2)$, 每个根均出现 2 次, 因而是一分圆多项式的平方. □

注167. 对 Graeffe 检测方法我们这里尚需说明一点, 由于 Graeffe 过程产生的多项式 f_1 其首项系数为正, 故在算法前 3 步三个判断中我们需使相应的多项式首项系数也为正, 即第 1 步中的 f, 第 2 步中的 $f(-x)$ 和第 3 步中的 f_2.

下面我们举例说明算法.

例12.2. 考虑 $f = x^8 - x^7 + x^5 - x^4 + x^3 - x + 1$.

解: 由于 $f = x^8 - x^4 + 1 + x(-x^6 + x^4 + x^2 - 1)$, 则

$$f_1(x) = (x^4 - x^2 + 1)^2 - x(-x^3 + x^2 + x - 1)^2 = x^8 - x^7 + x^5 - x^4 + x^3 - x + 1 = f(x),$$

故 f 是分圆多项式. 事实上 $f = \Phi_{15}$. ◇

例12.3. 我们再举一个例子, 取 $f = x^8 - x^6 + x^4 - x^2 + 1$.

解:

$$f_1(x) = (x^4 - x^3 + x^2 - x + 1)^2 = f_2^2,$$

而对于 $f_2 = (x^4 + x^2 + 1) + x(-x^2 - 1)$, 可得

$$f_3 = (x^2 + x + 1)^2 - x(-x - 1)^2 = x^4 + x^3 + x^2 + x + 1 = f_2(-x)$$

是分圆多项式, 综上 f 是分圆多项式. 事实上 $f = \Phi_{20}$. ◇

12.6.3 Euler 反函数方法

细心的读者可能已经注意到只用上节所说的方法虽然能够检测出多项式是否为分圆多项式, 但不能判断其阶数. 如果是为了精确求解方程, 那么我们仍需知道其阶数 n. 本节介绍的方法可以解决这一问题.

Euler 反函数方法本质上是简单的. 假设 f 是一 d 次不可约多项式, 倘若其为 n 阶分圆多项式, 那么我们有 $d = \phi(n)$, 且 $f | x^n - 1$. 因此一个比较朴素的想法就是列举可能的 n, 再进行试除. 为了列举所有可能的 n 值, 我们给出对函数 $\phi(n)$ 的一个估计:

定理12.11 (Euler 函数的估计)**.** *对于函数 $\phi(n)$, 我们有如下估计([37]247 页):*

1. $n \le 3\phi(n)^{3/2}, \quad \forall n \ge 2,$
2. *直接计算可得* $n \le 5\phi(n), \quad \forall n < 3000.$

除了利用试除法检测, 我们还可以通过提升多项式根的幂次来检测. 即对于 n 阶的分圆多项式, 其根经过 n 次幂后, 必为 1. 由上一小节的 Graeffe 过程我们可以知道, Graeffe 多项式

$$f_1 = \text{graeffe}(f)$$

的所有根恰为 f 的根的平方. 事实上通过下面的定理, 我们可以利用结式来计算 n 阶 Graeffe 多项式

$$f_1 = \text{graeffe}_n(f),$$

使得其所有的根恰为 f 的根的 n 次幂.

定理12.12 (n 阶 Graeffe 多项式)**.**

$$\text{graeffe}_n(f(x)) = \text{res}_y(f(y), y^n - x).$$

例12.4. 仍然考虑 $f = x^8 - x^7 + x^5 - x^4 + x^3 - x + 1$.

解: $d = \deg f = 8$, 于是 $n < 5d = 40$, 经检验发现 $f | x^{15} - 1$, 因此 $f = \Phi_{15}$.

如果不用试除法, 则通过计算 n 阶 Graeffe 多项式可以知道

$$\text{graeffe}_{15}(f) = (1 - x)^8,$$

于是也可得到 $f = \Phi_{15}$. ◇

12.6.4 位移分圆多项式检测

设 $f(x)$ 是一个分圆多项式, m 为一整数, 则我们如何才能检测出如 $f(x + m)$ 的形式呢?

首先我们注意到分圆多项式的常数项均为 ± 1, 因此对于给定的任何一个整系数多项式 $f(x)$, 若要找到 m 使 $f(x + m)$ 是分圆多项式, 则必有 $f(m) = \pm 1$. 因此我们要求解方程 $f(x) \pm 1 = 0$ 的整数根, 这可以用有限域因子分解算法一章中提到的算法, 也可以取 $f(x) \pm 1$ 的常系数的所有整数因子来尝试其是不是该方程的根.

下面给出一个具体的例子来说明这一方法.

例12.5. 考虑 $f = x^8 + 16x^7 + 111x^6 + 436x^5 + 1061x^4 + 1640x^3 + 1575x^2 + 860 + 205$.

解: 可以求出 $f(x) + 1 = 0$ 无整根, 而 $f(x) - 1 = 0$ 有整根 -1, -2, -3, 其中将 -2 代入可得

$$f(x - 2) = x^8 - x^6 + x^4 - x^2 + 1,$$

正是分圆多项式 Φ_{20}. ◇

12.7 (一元)复合函数分解

12.7.1 复合函数分解算法

这一小节我们来处理一些复合函数分解的问题. 函数复合我们已经很了解了, 即对于两个多项式 g, h, 我们可以求出它们的复合函数 $f = g \circ h = g(h)$. 现在我们要考虑的是它的逆问题, 即对于给定的 f, 能不能找到这样的 g 和 h, 以及如何找到他们.

为什么要考虑这个问题呢?我们知道, 对于代数方程精确求解来说, 我们已知能精确求解的有低于 4 次的多项式以及前面所述的分圆多项式, 事实上, 某些高于 4 次的多项式, 如

$$f = x^6 + 2x^3 + 1,$$

也可以看作 2 次方程来精确求解, 这里就要用到复合函数分解的算法. 我们容易看出 $f = g \circ h$, 其中 $g = x^2 + 2x + 1$, $h = x^3$.

本节内容可参考 [173].

我们来考虑一元情形. 设一元 n 次多项式 $f \in F[x]$, 对于 n 的一个因子 $r > 0$, 令 $s = n/r$, 我们要找到多项式 $g, h \in F[x]$ 使得它们的次数分别为 r, s 且 $f = g \circ h = g(h)$. 我们这里考虑非病态的情形(tame case, 见 [173]), 即域 F 的特征 $p = \mathrm{char}(F)$ 不整除 r.

我们有下面的关于复合函数分解的唯一性定理:

定理12.13 (Ritt 第一定理)**.** *完全分解(complete decomposition)$f = f_1 \circ f_2 \circ \cdots \circ f_k$(其中 $f_1, \ldots, f_k$ 均为不可分解的)在不计以下变换的意义下是唯一的:*

$\forall f \in F[x]$, $c, d \in F (c \neq 0)$, $r, m \geq 2$,

1. $f \circ (cx + d) \circ ((x - d)/c) = f$.
2. $(x^m \cdot f^r) \circ x^r = x^r \circ (x^m \cdot f(x^r))$.
3. $T_r \circ T_m = T_m \circ T_r = T_{rm}$, *其中 T_i 是 i 阶 Chebyshev 多项式.*

注168. Chebyshev 多项式的定义为

$$T_n(x) = \cos(n \cos^{-1} x),$$

于是有

$$T_r \circ T_m = \cos(r \cos^{-1}(\cos(m \cos^{-1} x))) = \cos(rm \cos^{-1} x) = T_{rm}.$$

考虑到分解在上述变换下的不定性, 下面我们设 $f = g \circ h$, 且 $a = \mathrm{lc}(f)$, $c = \mathrm{lc}(h)$, 于是我们有

$$\frac{f}{a} = \left(\frac{1}{a} g(cx + h(0))\right) \circ \frac{h - h(0)}{c},$$

这相当于将一个首一多项式分解为两个首一多项式的复合, 并且第二个多项式(h)常数项为 0. 考虑下面给出的定义:

定义12.7. 记 M 为 $F[x]$ 中所有首一多项式的集合, 设有 $f \in M$, 定义如下集合

$$\mathrm{DEC}_{n,r}^F = \{(f, (g, h)) \in M \times M^2 | f = g \circ h, \deg f = n, \deg g = r, h(0) = 0\},$$

称为 f 的分解问题的解.

下面在提出分解算法之前, 为了方便先给出一个定义:

定义12.8 (The reversal of f). 设 $f = x^n + a_{n-1}x^{n-1} + \cdots + a_0 \in F[x]$, 记 $\tilde{f} = a_0x^n + \cdots + a_{n-1}x + 1 = x^n f(1/x)$.

算法12.8 (一元复合函数分解).

输入:首一多项式 $f \in F[x]$, 其次数 $n = rs$, 并且 $\mathrm{char}(F) \nmid r$,

输出:$\mathrm{DEC}_{n,r}^F$.

1. 计算 $\tilde{h} \in F[x]$, $\deg \tilde{h} < s$ 且 $\tilde{h}^r \equiv \tilde{f} \bmod x^s$, $\tilde{h}(0) = 1$, 令 $h = x^s\tilde{h}(1/x) \in F[x]$,
2. 计算"Taylor 展开"的系数 $b_0, \ldots, b_r \in F[x]$ 如下:
$$f = \sum_{0\le i\le r} b_i h^i, \quad \deg b_i < \deg h (\forall i).$$
3. 若 $b_0, \ldots, b_r \in F$, 令 $g = \sum_{0\le i\le r} b_i x^i \in F[x]$, 并输出 $(f, (g, h))$ 终止, 否则输出 $\varnothing$ 终止.

算法有效性. 由 $\tilde{h}(0) = 1$ 知 $\mathrm{lc}(h) = 1$, 由 $\deg \tilde{h} < s$ 知 $h(0) = 0$. 反过来设 $f = g \circ h$, 并且满足相应条件, 则 f 和 h^r 的最高 s 项相同, 即 $\deg(f - h^r) \le n - s$. 仍记 $\tilde{h} = x^s h(1/x)$, 于是 $x^n h(1/x)^r = (x^s h(1/x))^r = \tilde{h}^r$, 且

$$\deg(f-h^r) \le n-s \Leftrightarrow x^n((f-h^r)(1/x)) \equiv 0 \pmod{x^s} \Leftrightarrow \tilde{f}-\tilde{h}^r \equiv 0 \pmod{x^s}.$$

证毕. □

注169. 利用相关的快速算法([173]283 页事实 2.1), 则整个算法的复杂度为 $O(M(n)\log n)$, 其中 $M(n)$ 是两个次数为 n 的多项式相乘所需的代数运算. Taylor 展开只需由 Euclid 除法一步步计算即可. 算法中要用到的多项式开方算法等一些需要补充的问题将在下一小节介绍. 该算法结果的唯一性是由于开方得到的常数项为 1 的 r 次根 $\tilde{h}$ 唯一.

进而我们有下面的推论:

推论12.4. *1. 设有两个分解 $f = g_1 \circ h_1 = g_2 \circ h_2$ 且 $\deg g_1 = \deg g_2 = r$, 则两个分解是相似的, 即它们之间可以通过三个变换中的仿射变换相联系:$\exists c, d \in F (c \ne 0)$, 使得 $g_1 = g_2(cx + d)$, $h_1 = (h_2 - d)/c$.*

2. $\#\mathrm{DEC}_{n,r}^F \le 1$.

3. 设 k 是 F 的某个扩域, 且 $(f,(g,h)) \in \mathrm{DEC}_{n,r}^k$, $h = cx^s + \cdots + d \in k[x]$, 则 $g_1 = g(cx+d) \in F[x]$, $h_1 = (h-d)/c \in F[x]$, 且 $(f,(g_1,h_1)) \in \mathrm{DEC}_{n,r}^F$.

利用复合分解算法, 我们可以进行一元多项式的完全复合分解.

算法12.9 (完全复合分解).

输入:首一 n 次多项式 $f \in M \subset F[x]$, 且 $\mathrm{char}(F) \nmid n$,

输出:f 的完全复合函数分解.

1. 计算整数 n 的因子分解 $n = p_1^{e_1} \cdots p_k^{e_k}$, 令 $d(n) = (e_1+1)\cdots(e_k+1)$ 为 n 的正因子个数, 且记 $r_1 = 1 < r_2 < \cdots < r_{d(n)} = n$ 为其正因子,

2. 对 j 从 2 循环到 $d(n)-1$, 求解问题 DEC_{n,r_j}^F, 对于寻找到的第一个解 $(f,(g,h))$, 递归调用本算法求解 h 的分解问题, 得到分解 $h = f_2 \circ f_3 \circ \cdots \circ f_k$,

3. 输出 $(f_1,\ldots,f_k)$.

12.7.2 形式幂级数的一些基本操作

[104] 中对形式幂级数的一些基本的算术作了介绍.

定义12.9. $f = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n + \cdots$ 称为形式幂级数(formal power series), 其中系数 a_i 在域中.

我们可以考虑有限域或复数域上的形式幂级数, 这里我们假定都是在 $\mathbb{C}$ 上讨论. 虽然一般计算机上是无法表达无穷项的形式幂级数, 好似浮点数表示的实数都是有限精度的, 但我们仍有讨论它们的必要. 一般而言, 我们也只考虑形式幂级数的前若干项. 这样, 问题就化为了在模 $x^N (N \in \mathbb{N})$ 下的多项式算术问题.

首先, 我们可以得到形式幂级数的乘法算法:

算法12.10 (形式幂级数乘法算法).

输入:形式幂级数 $g = \sum_{0\le i\le\infty} g_ix^i$, $h = \sum_{0\le i\le\infty} h_ix^i$,

输出:$f = gh$ 的第 n 次幂的系数 f_n.

$$f_n = \sum_{k=0}^{n} g_k h_{n-k}.$$

由上面算法, 我们可以很方便地得到除法算法:

算法12.11 (形式幂级数除法算法).

输入:形式幂级数 g, h, 各符号意义同上, 其中 $h_0 \neq 0$,

输出:$f = g/h$.

由

$$f_n = \left(g_n - \sum_{k=0}^{n-1} f_k h_{n-k} \right) / h_0$$

依次可得 $f_0, f_1, \cdots$

下面考虑求幂算法, 此算法可用于前文复合函数分解中所依赖的开方. 考虑一个幂级数 $g(x)$, 对于某个实数 α, 欲求级数 $f(x) = g(x)^\alpha$, 首先我们可设 $g(x)$ 有如下形式:

$$g(x) = g_m x^m \left(1 + \frac{g_{m+1}}{g_m} x^{m+1} + \cdots \right),$$

则其 α 次幂为:

$$f(x) = g_m^\alpha x^{m\alpha} \left(1 + \frac{g_{m+1}}{g_m} x^{m+1} + \cdots \right)^\alpha,$$

由上式可以看出问题归结为一个常数项为 1 的形式幂级数的幂次计算, 下面给出两种方法. 设 $g(x) = 1 + h(x)$, 其中 $h(0) = 0$, 利用二项式定理展开, 我们可以可得:

$$g(x)^\alpha = (1 + h(x))^\alpha = 1 + \binom{\alpha}{1} h(x) + \cdots + \binom{\alpha}{n} h(x)^n + \cdots.$$

另一种方法由 Euler 发现, 由 $f(x) = g(x)^\alpha$, 我们求其微分可以得到:

$$f'(x) = \alpha g(x)^{\alpha-1} g'(x),$$

亦即

$$f'(x) g(x) = \alpha f(x) g'(x),$$

若将其系数展开, 并取 x^{n-1} 项的系数可得等式:

$$\sum_{k=0}^{n} k f_k g_{n-k} = \alpha \sum_{k=0}^{n} (n-k) f_k g_{n-k},$$

于是

$$\begin{aligned} f_n &= \sum_{k=1}^{n} \left(\frac{\alpha+1}{n} k - 1 \right) g_k f_{n-k} \\ &= ((\alpha + 1 - n) g_1 f_{n-1} + (2\alpha + 2 - n) g_2 f_{n-2} + \cdots + n\alpha g_n f_0)/n. \end{aligned}$$

从上式可以看出, Euler 给出的算法复杂度为 $O(n^2)$, 一般地, 我们采用上面的二项式展开算法或 Euler 给出的方法即可, 但若要追求效率, [173] 中提到该算法复杂度可达 $O(M(n)\log r)$, 其中 n 是考虑的项数, r 是开方次数($\alpha = 1/r$ 情形), 此由文献 [40] 给出的 Newton 迭代等算法可以达到. 关于形式幂级数操作的快速算法可以参考该文, 另外对于形式幂级数的求逆也可参考 [108] 等文献, 这里就不再赘述了.

13 代数方程组求解

本章名为代数方程组的求解, 所以我们将从求解代数方程组的角度来引出一系列概念和算法. 根据我们从解线性方程组得到的经验, 我们需要发展一套消元方法, 以使将方程组化为类似于三角阵的形状, 这样引出两种算法. 一是吴文俊院士于上世纪七十年代为实现几何定理机器证明而提出的吴方法, 它不仅能进行机器证明, 同时特征列是一种三角列, 也可以用于代数方程组的求解. 另外一种是 Gröbner 基方法, 它除了将多项式组化为三角形的基可用于解方程外, 还可用于多项式理想的计算等.

如果只从几何证明的角度, 吴方法无疑比 Gröbner 基要高效很多, 正是如此, 吴方法在几何证明领域中在世界上都是十分有名的. 然而, 吴方法毕竟相当于一种不完全的约化, 其除法用的是伪除法, 因而在解方程, 多项式理想计算等方面不如 Gröbner 基方法.

另外, 由高等代数学 [12] 我们知道还可以利用结式的计算来进行消元, 这一过程及其理论本身比较简单, 因此本章第一节我们先介绍结式消元法, 第二节和第三节分别介绍吴方法和 Gröbner 基.

在介绍消元方法之前, 我们简单说明一下多元方程组求解的思路. 首先经过消元方法, 将问题化为一元方程, 利用上一章介绍的各种数值方法和精确求解方法可以分别求解. 这里值得一提的是如果要求精确解, 设 n 个变元为 $x_1, x_2, \ldots, x_n$, 一般我们将方程组消元 n 次, 每次化为只含其中某一变元 x_i 的一元方程, 以此求出 x_i 的精确解. 实际中, 我们有可能对每个 x_i 都求出了一系列的解, 剩下的任务是将它们搭配成 n 元序对, 即原方程组的解. 如果是根式解或单位根, 我们可以很容易

地用一定精度的数值方法来搭配. 如果没有根式解或不是单位根, 一般情况下我们表示代数数的方法是用极小多项式 f(或者化零多项式)和 i 表示 f 的第 i 个根(我们可以自己规定一种根的排序). 于是配对的过程可以用数值方法, 即在一定精度数值求解此根, 或者用区间隔离的方法, 将区间分得足够小来逼近根, 用区间算术代替根的运算以搭配. 我们举一个简单的例子来说明:

$$\begin{cases} x^2+y^2-1 & =0 \\ x^2-2x+y^2-2y+1 & =0 \end{cases}$$

消元方法给出 $x^2-x=0$ 和 $y^2-y=0$, 因此 $x=0,1$ 且 $y=0,1$, 我们需要将 2×2 共四种情况进行搭配, 当然, 这里举的例子很简单, 更多的情况下解为复杂的根式或者为一高次不可约方程的某个根. 我们很容易搭配出两组解 $(x,y)=(0,1)$ 或 $(1,0)$.

13.1 结式

我们所说的结式一般都是指 Sylvester 结式, 利用结式进行消元基于我们以前提过的一个命题(见推论 8.3), 现在重述如下:

定理13.1. *设 R 是 UFD, 且有非零多项式 $f,g\in R[x]$, 则 $\gcd(f,g)$ 非平凡当且仅当 $\operatorname{res}(f,g)=0$.*

现在我们考虑一个二元多项式方程组. 设 $f,g\in\mathbb{C}[x,y]=\mathbb{C}[y][x]$, 我们可以把它们看成 $\mathbb{C}[y]$ 上关于 x 的一元多项式. 那么利用上面的定理, 我们有 $\gcd_x(f,g)$ 非平凡当且仅当 $\operatorname{res}_x(f,g)=0$. 现在我们要求它们的公共根, 不妨设 x_0 是其一根, 则有 $(x-x_0)|\gcd_x(f,g)$, 故有关于 y 的方程

$$\operatorname{res}_x(f,g)=0.$$

由此, 我们实际上进行了消元, 将方程组化为一个只含有 y 的方程.

事实上, 我们可以将这种消元方法推广到 n 个变元的情形. 设变元分别为 $x_1,x_2,\ldots,x_n$, 则我们首先可以从 n 个方程中取出 $(n-1)$ 对方程, 两两用结式法消去 x_1, 之后得到 $(n-1)$ 个关于 $x_2,\ldots,x_n$ 的方程, 再将此过程递归做下去, 最终可求出方程的解.

例13.1. 考虑多项式方程组

$$\begin{cases} f_1(x,y,z)=x^2+y^2+z^2-1=0, \\ f_2(x,y,z)=x+y+z=0, \\ f_3(x,y,z)=x^2-2x+y^2-2y+z^2+2z=0. \end{cases}$$

计算得

$$f_4 = \mathrm{res}_z(f_1, f_2) = 2x^2 + 2xy + 2y^2 - 1,$$

$$f_5 = \mathrm{res}_z(f_2, f_3) = 2x^2 + 2xy + 2y^2 - 4x - 4y,$$

可见变元 z 已被消去, 再计算上面二式关于 y 的结式可将 y 消去:

$$f_6 = \mathrm{res}_y(f_4, f_5) = 4(16x^2 - 4x - 7).$$

于是得到一元方程 $16x^2 - 4x - 7 = 0$.

13.2 吴方法

13.2.1 一些基本概念

吴方法也称为特征列方法, 这原本是由 J. F. Ritt 在他的关于微分代数的工作中引入, 上个世纪 70 年代末, 吴文俊在建立几何定理机器证明时大大发展了这一领域. 它同 Gröbner 基类似, 也是一种消元方法, 在几何定理证明, 代数方程求解等方面均有很重要的应用. 关于吴方法, 可以参考 [14], [13], 也可见吴文俊本人的著作 [3]4.3 节.

下面设 k 是一特征为 0 的域, $k[X] = k[x_1, \ldots, x_n] = R$, 并取字典序为 $x_1 < x_2 < \cdots < x_n$.

定义13.1. 对于单项式 t, 记 t 中含 x_i 最大的下标 i 为 p, 定义它的类 $\mathrm{cl}(t) = p$. 即

$$\mathrm{cl}(t) = p = \max\{i : x_i | t\},$$

其中非 0 常数的类定义为 0.

定义13.2. 对于多项式 $f \in R$, 定义其类为 $\mathrm{cl}(f) = \mathrm{cl}(\mathrm{lt}(f))$.

定义13.3. 定义一种全序关系 $<_C \subset R \times R$, 对于 $f, g \in R$, 称 $f < g$ 如果下列条件之一满足:

1. $\mathrm{cl}(f) < \mathrm{cl}(g)$,
2. $\mathrm{cl}(f) = \mathrm{cl}(g) = p > 0 \wedge \deg_p f < \deg_p g$.

若 $f \not< g \wedge g \not< f$, 则称 f 和 g 级别相同, 记作 $f \sim g$.

注170. 若 $f < g$, 则 $\mathrm{lt}(f) < \mathrm{lt}(g)$. 反之则不然, 例如 $f = x_1x_2^2x_3$, $g = x_1^2x_3$ 级别相同.

注171. 这样的序不仅是全序, 而且是良序.

定义13.4. 设 $\mathrm{cl}(f) = p > 0$, 若 $\deg_p g < \deg_p f$, 则称 g 对 f 是约化的, 记为 $g(\mathrm{red}\, f)$.

注172. 若 $g < f$, 则显然 $g(\mathrm{red}\, f)$. 但当 $g(\mathrm{red}\, f)$ 时, 不一定有 $g < f$. 例如 $g = x_1x_3$, $f = x_2$.

下面记 $\mathrm{cl}(f) = p > 0$, 且

$$f = f_0x_p^m + f_1x_p^{m-1} + \cdots + f_m,$$

$$g = g_0x_p^M + g_1x_p^{M-1} + \cdots + g_M,$$

首先

定义13.5. $f_0 \in k[x_1, x_2, \ldots, x_{p-1}, x_{p+1}, \ldots, x_n]$ 称为 f 的初式.

当 g 相对 f 不约化时, 即 $M > m$, 我们可由下面定义的伪除法将其约化.

定义13.6. $\exists s \in \mathbb{N}$ 使得 $f_0^s g = qf + r$, 其中 $r(\mathrm{red}\, f)$. 将 g 化为 r 的过程称为约化, 此处 s 宜取 $M - m + 1$.

13.2.2 升列

我们引入一系列的定义.

定义13.7. 多项式序列 $F = \{f_1, f_2, \ldots, f_r\}$ 称为升列(ascending set), 若下列条件之一满足:

1. $r = 1$ 且 $f_1 \neq 0$,
2. $r > 1$, 且 $0 < \mathrm{cl}(f_1) < \mathrm{cl}(f_2) < \cdots < \mathrm{cl}(f_r)$, 且 $\forall j > i$, 有 $f_j(\mathrm{red}\, f_i)$.

由升列的定义我们知道升列的项数 r 必然不大于未定元个数 n.

定义13.8. 升列称为矛盾列, 如果升列只由一个非零常数组成. 矛盾列构成的多项式方程组是无解的.

定义13.9. 设 F 是升列, 对多项式 $g \in R$, 若 $\forall f \in F$ 有 $g(\mathrm{red}\, f)$, 则称 g 相对于 F 是约化的, 记为 $g(\mathrm{red}\, F)$.

定义13.10. 定义升列上的全序关系 $\prec$, 对于两个升列 $F = \{f_1, \ldots, f_r\}$, $G = \{g_1, \ldots, g_s\}$, 称 $F \succ G$, 若下列条件之一满足:

1. $\exists j \leq \min(r,s)$, 使得

$$f_1 \sim g_1, f_2 \sim g_2, \ldots, f_{j-1} \sim g_{j-1}, f_j > g_j,$$

2. $s > r$ 且 $f_1 \sim g_1, \ldots, f_r \sim g_r$.

若 $F \not\prec G \wedge G \not\prec F$, 则称二者级别相同, 记为 $F \sim G$.

注173. 若 $F \sim G$, 显然有 $r = s$ 且 $f_i \sim g_i (\forall i)$.

下面的定理很重要, 是特征列方法的基础.

定理13.2 (Ritt 引理)**.** *设 $F_1, F_2, \ldots, F_q, \cdots$ 是不增升列的序列, 即 $\forall q$, 有 $F_{q+1} \prec F_q \vee F_{q+1} \sim F_q$, 则 $\exists q'$, $\forall q > q'$, 有 $F_q \sim F_{q'}$, 即有极小升列 $F_{q'}$.*

证明. 记 $r_q = \#F_q$, $f_q \in F_q$ 是每个升列中第一个多项式, 则

$$f_1, f_2, \ldots, f_q, \cdots$$

是一个不增列. 对于良序下的不增列我们显然有 $q_1 \in \mathbb{N}$ 使得 $\forall q > q_1$ 时, $f_{q_1} \sim f_q$.

我们再考虑 $q \geq q_1$ 时 F_q 中第 2 个多项式构成的序列

$$f_{q_1}^{(1)}, \ldots, f_q^{(1)}, \cdots,$$

同样的分析给出 $q_2 \in \mathbb{N}$ 使得 $q \geq q_2$ 时有 $f_q^{(1)} \sim f_{a_2}^{(1)}$. 由于每个升列都有 $r_q \leq n$, 我们将这个过程继续下去总会终止, 于是可找到 $q' \in \mathbb{N}$, 使得 $q \geq q'$ 时, $r_q = r_{q'}$ 且 $F_q \sim F_{q'}$. □

推论13.1. *严格下降的升列序列必为有限的.*

13.2.3 基本列

设 A 是一有限非零多项式的集合, 其中必含有升列, 我们记从中选出的升列的全体集合为 $AS(A)$.

定义13.11. 若 $F \in AS(A)$ 是当中的一个极小元, 则称 F 是 A 的基本列(basic set).

定理13.3. *A 上的基本列存在, 且能在有限步内构造出来.*

证明. 我们只需给出构造性的证明即可. 设 $A_1 = A$.

先取 f_1 为 A_1 中的某个最小元, 若 $\mathrm{cl}(f_1) = 0$ 则 $\{f_1\}$ 已是基本列. 若 $\mathrm{cl}(f_1) > 0$, 记 $B_1 = A_1 \setminus \{f_1\}$. 若 B_1 中元素对 f_1 均未约化, 则 $\{f_1\}$ 仍然是基本列, 否则记 B_1 中对 f_1 约化的多项式全体为 A_2.

$\forall f_2 \in A_2$, 显然有 $f_2 \geq f_1$. 倘若 $f_2 \sim f_1$, 则与 $f_2(\mathrm{red}\, f_1)$ 矛盾. 故 $f_2 > f_1$, 由于 $f_2(\mathrm{red}\, f_1)$, 则只能 $\mathrm{cl}(f_2) > \mathrm{cl}(f_1)$.

再取 f_2 为 A_2 中的某个最小元, 若 $B_2 = A_2 \setminus \{f_2\}$ 中元素关于 f_2 都未约化, 则 $\{f_1, f_2\}$ 已是基本列, 否则可以构造出 $A_3 \cdots$

由于 $f_1, f_2, \cdots$ 的类是严格增加的, 则上述过程必能有限步终止, 得到基本列. □

鉴于基本列的存在性, 我们记集合 A 的全体基本列的集合为 $BS(A)$. 定理 13.3 的构造性证明过程实际上已给出了求多项式集合 A 的基本列的方法.

定理13.4. *A 是由非零多项式构成的有限集, $F = \{f_1, \ldots, f_r\} \in BS(A)$, $\mathrm{cl}(f_1) > 0$, 设 g 是一非零多项式, 且 $g(\mathrm{red}\, f_i)(\forall f_i \in F)$, 设 $B = A \bigcup \{g\}$, 则 $\exists G \in BS(B), G \prec F$.*

证明. 若 $\mathrm{cl}(g) = 0$, 则 $G = \{g\}$ 满足要求.

若 $\mathrm{cl}(g) = p > 0$, 则 $\exists s (1 \leq s \leq r)$ 使得 $\mathrm{cl}(f_{s-1}) < p \leq \mathrm{cl}(f_s)$, 又由于 $g(\mathrm{red}\, f_s)$, 则 $g < f_s$, 故 $G = \{f_1, f_2, \ldots, f_{s-1}, g\}$ 满足要求.

若 $\mathrm{cl}(g) > \mathrm{cl}(f_r)$, 则 $G = \{f_1, \ldots, f_r, g\}$ 满足要求. □

13.2.4 特征列与解方程

定义13.12. 若 g 对升列 $F = \{f_1, \ldots, f_r\}$ 不是约化的, 设 I_i 为 $f_i \in F$ 的初式, 则有带余除法:

$$I_r^{s_r} g = Q_r f_r + R_r, \quad R_r(\mathrm{red}\, f_r),$$

$$I_{r-1}^{s_{r-1}} R_r = Q_{r-1} f_{r-1} + R_{r-1}, \quad R_{r-1}(\mathrm{red}\, f_{r-1}),$$

$$\cdots$$

$$I_1^{s_1} R_2 = Q_1 f_1 + R_1, \quad R_1(\mathrm{red}\, f_1).$$

显然 R_1 对 F 是约化的, 我们称其为 g 对 F 的余式, 记为 $g \,\mathrm{rem}\, F$.

下面给出特征列的定义:

定义13.13. 设 P 为由非零多项式构成的有限集, 称升列 $C \in AS(P)$ 为其特征列(characteristic set), 如果 $C \subset \langle P \rangle$ 且 $\forall p \in P$ 有 $p \,\mathrm{rem}\, C = 0$.

算法13.1 (求特征列的算法).

输入:非零有限多项式集合 P,

输出:P 的特征列 F.

1. $P_1 = P$, $i = 1$,
2. 求出 P_i 的一个基本列 $F_i \in BS(P_i)$,
3. 将 $P_i \setminus F_i$ 中的多项式对 F_i 求余式, 其非零项构成集合 R_i,
4. 若 $R_i \neq \varnothing$, 令 $P_{i+1} = P_i \bigcup R_i$, $i = i + 1$, 转 2 步,
5. 输出 F_i.

对于上面的算法过程, 设其在 $i = m$ 时终止, 我们很容易有如下断言:

$$V(P_1) = V(P_2) = \cdots = V(P_m).$$

而对于算法的终止性, 由定理 13.4 可知我们得到如下的递降基本列:

$$F_1 \succ F_2 \succ \cdots$$

由于 Ritt 引理, 我们可以断言在某一步, 例如 $i = m$ 时, 算法会终止, 即此时会得到 $R_i = \varnothing$.

关于特征列和多项式零点的关系, 有如下的定理:

定理13.5. *设 $F = \{f_1, f_2, \ldots, f_r\}$ 为求得的基本列, 且 f_i 的初式为 I_i, $J = I_1 I_2 \cdots I_r$, 则 $V(P) = (V(F) \setminus V(J)) \bigcup V(P, I_1) \bigcup V(P, I_2) \bigcup \cdots \bigcup V(P, I_r)$.*

证明. 由伪除法的定义, 我们知道对于多项式 $p \in P$, 由于 $p(\operatorname{red} F)$, 则存在 $q_r, \ldots, q_1$ 使得

$$I_r^{s_r} \cdots I_1^{s_1} p = q_r f_r + \cdots + q_1 f_1,$$

由此等式显然可以看出右边任何一项都是左边的子集, 因此右 $\subset$ 左. 现在 $\forall a \in V(P)$, 若 $a \in V(J)$, 则必有某个 i 使得 $a \in V(I_i)$, 此时易知 $a \in V(P, I_i)$, 若 $a \notin V(J)$, 则 $a \in V(F) \setminus V(J)$, 因此有左 $\subset$ 右. □

注174. 若 F 是矛盾列, 则有 $V(P) \subset V(F) = \varnothing$.

定理13.6. *对于有限非零多项式集合 P, 存在一系列的特征列 $F_1, F_2, \cdots$ 使得*

$$V(P) = \bigcup [V(F_i) \setminus V(J_i)],$$

其中 J_i 为 F_i 中各多项式初式之积.

证明. 根据前面定理, 我们已有分解

$$V(P) = (V(F) \setminus V(J)) \bigcup (\bigcup V(P, I_i)).$$

首先我们证明若 $I_i \neq 0$, 则 $I_i(\operatorname{red} P)$. 由于 I_i 是特征列 F 中 f_i 的初式, 则显然有 $I_i(\operatorname{red} F)$. 因此 $I_i(\operatorname{red} P)$.

再用递归思想, 对每个不为零的 I_i, 分解 $V(P, I_i)$, 若其特征列不是矛盾列, 则可得到递减的特征列序列, 由定理 13.4 可知其必有限终止, $V(P)$ 可化为本定理形式. □

13.3 Gröbner 基

Gröbner 基是 Buchberger 于 1965 年在其博士毕业论文中提出, 最初是用来解决多项式方程组的问题. 其后经过发展, 它在多元多项式环的理想等问题上也有重要的应用. [17] 一书对此有详细介绍, 另外 [174], [13] 对此也有介绍.

13.3.1 一些概念与介绍

我们主要是为了处理多元多项式而引入 Gröbner 基, 为了方便起见, 我们先给出一些记号上的说明.

设 F 为一域, 以 X 表示 n 个不定元 $x_1, x_2, \ldots, x_n$, 则记多项式环 $R = F[X] = F[x_1, x_2, \ldots, x_n]$, 设有 s 个多项式 $f_1, \ldots, f_s \in R$, 由它们生成的理想记作

$$I = \langle f_1, f_2, \ldots, f_s \rangle = \left\{ \sum_{1 \leq i \leq s} q_i f_i | q_i \in R \right\}.$$

定义13.14. 对于上面的理想 I, 定义其仿射簇为

$$V(I) = \{a = (a_1, a_2, \ldots, a_n) \in F^n | f_i(a) = 0, i = 1, 2, \ldots, s\}.$$

显然我们有 $V(f_1, f_2, \ldots, f_s) = \bigcap_{i=1}^{s} V(f_i)$, 且 $\forall f \in I(f(V(I)) = 0)$.

采用上面的记号, $f_1, \ldots, f_s$ 显然是 I 的基, 我们知道, 在一元多项式环中, 由于其是主理想环, 我们有

$$\langle f_1, \ldots, f_s \rangle = \langle \gcd(f_1, \ldots, f_s) \rangle = \langle g \rangle,$$

且对于任何一个多项式 f, 将其对 g 作 Euclid 除法得到 $f = qg + r$, 则 $f \in \langle g \rangle \Leftrightarrow r = 0$. 但对于多元情形, 这些良好的性质未必成立, 比如

$$\langle x, y \rangle \neq F[x, y] = \langle 1 \rangle = \langle \gcd(x, y) \rangle.$$

对于指标 $\alpha=(\alpha_1,\alpha_2,\ldots,\alpha_n)\in\mathbb{N}^n$, 定义 $X^\alpha=x_1^{\alpha_1}x_2^{\alpha_2}\cdots x_n^{\alpha_n}$, 称为单项式(Monomial), 将全体单项式集合记作 $M\subset R$.

我们可以定义 M 中的一种良序 $<$, 使其满足与加法的和谐性, 即对任何三个指标 $\alpha,\beta,\gamma\in\mathbb{N}^n$, 有 $\alpha<\beta\Rightarrow\alpha+\gamma<\beta+\gamma$. 下面给出几个序的例子:

例13.2. M 上的字典序(Lexicographic order) $<_{lex}$: $\alpha<_{lex}\beta\Leftrightarrow\alpha-\beta$ 左边第一非零分量为负.

例13.3. M 上的分级字典序(Graded lexicographic order)$<_{lex1}$:$\alpha<_{lex1}\beta\Leftrightarrow\|\alpha\|<\|\beta\|\vee(\|\alpha\|=\|\beta\|\wedge\alpha<_{lex}\beta)$, 其中 $\|\cdot\|$ 是 1-范数.

例13.4. M 上的分级逆字典序(Graded reverse lexicographic order)$<_{lex2}$:$\alpha<_{lex2}\beta\Leftrightarrow\|\alpha\|<\|\beta\|\vee(\|\alpha\|=\|\beta\|\wedge\alpha-\beta$最右非零分量为零).

我们一般取字典序即可, 就以 $<$ 来表示. 在该序下, 我们可以定义多项式 f 的领项 $\mathrm{lt}(f)$ 为 f 最大的单项式, 类似地可定义领项系数 $\mathrm{lc}(f)$ 和领项单项式 $\mathrm{lm}(f)$. 一个多项式次数的定义为 $\deg f=\deg\mathrm{lt}(f)\in\mathbb{N}^n$. 有了这些概念, 我们可以像在一元多项式环中那样做带余除法, 下面给出 R 中带余除法的算法:

算法13.2 (带余除法).

输入:$f,f_1,f_2,\ldots,f_s\in R$,

输出:$q_1,q_2,\ldots,q_s,r\in R$ 使得 $f=q_1f_1+\cdots+q_sf_s+r$ 且 r 中任何单项不被 $\mathrm{lt}(f_1),\ldots,\mathrm{lt}(f_s)$ 中任何一个整除, 即不可再约化.

1. $r=0$, $p=f$, $q_i=0(i=1,\ldots,s)$,
2. 当 $p\neq0$ 时, 循环做第 3 步,
3. 若存在某个 i 使 $\mathrm{lt}(f_i)|\,\mathrm{lt}(p)$ 则

$$q_i=q_i+\frac{\mathrm{lt}(p)}{\mathrm{lt}(f_i)},\quad p=p-\frac{\mathrm{lt}(p)}{\mathrm{lt}(f_i)}f_i,$$

   否则 $r=r+\mathrm{lt}(p)$, $p=p-\mathrm{lt}(p)$,
4. 输出 $q_1,q_2,\ldots,q_s,r$.

例13.5. 考虑 $f=x^2y+xy^2+y^2$, $f_1=xy-1$, $f_2=y^2-1$.

解: 用上面的算法计算除法, 我们发现, 第一步只可以用 f_1 来约化, 得到 $f=xf_1+(xy^2+x+y^2)$, 第二步我们可以用 f_1 或 f_2 来约化, 简单计算我们发现若这

一步用 f_1 来约化, 得到结果

$$f = (x+y)f_1 + f_2 + (x+y+1),$$

反之则得到

$$f = xf_1 + (x+1)f_2 + (2x+1),$$

我们看到, 约化的顺序不同会导致结果不同, 这也是多元多项式环区别于一元多项式环的性质之一. ◇

定义13.15. 在算法 13.2 第 3 步中若选取满足领项能整除 $\mathrm{lt}(p)$ 的最小的指标 i 对应的多项式进行约化, 则定义此时得到的 r 为余式 $f\,\mathrm{rem}(f_1, f_2, \ldots, f_s) = r$.

由前面约化结果的不唯一性我们知道, 并不能用余式是否为零来判断一个多项式是否在所考察的理想中, 为了解决种种在多元多项式环中出现的问题, 我们需要引入 Gröbner 基.

13.3.2 单项式理想及一些准备定理

单项理想即是指由一些单项式生成的理想, 若 A 是 N^n 的一个子集, 定义 $\langle x^A\rangle = \langle\{x^\alpha|\alpha \in A\}\rangle$.

引理13.1. $x^\beta \in I \Leftrightarrow \exists\alpha \in A(x^\alpha|x^\beta)$.

引理13.2. *设 I 是一个单项理想, 以下三个命题是等价的:*

(1)$f \in I$,

(2)f 中每个单项都属于理想 I,

(3)f 是 I 中某些多项式的 F-线性组合.

证明. (1)⇒(2) 设 $I = \langle x^A\rangle$, 则必有 $f = \sum_{\alpha\in A} q_\alpha x^\alpha$, 其中 q_α 是多项式. 由此可知 f 中每个单项必可被某个 x^α 整除.

(2)⇒(3)和(3)⇒(1)均显然. □

由引理第二个等价条件得到:

推论13.2. *两个单项理想 I_1, I_2 相等的充要条件是它们含有相同的单项式.*

定理13.7 (Dickson 引理)**.** $\forall A \subset \mathbb{N}^n, \exists$ 有限集 $B \subset A$ 使得 $\langle x^A\rangle = \langle x^B\rangle$.

证明. 为了便于证明, 我们引入 $\mathbb{N}^n$ 上的偏序 $\preccurlyeq$ 满足 $\alpha \preccurlyeq \beta \Leftrightarrow \forall i \in \{1, 2, \ldots, n\}(\alpha_i \leq \beta_i)$. 由此我们知道 $\alpha \preccurlyeq \beta \Leftrightarrow x^\alpha | x^\beta$.

设 B 为 A 的极小元集合, 即 $B = \{\beta \in A | \forall \alpha \in A(\alpha \nprec \beta)\}$.

于是 $\forall \alpha \in A, \exists \beta \in B(\beta \preccurlyeq \alpha)$, 如若不然, 首先 $\beta \neq \alpha \Rightarrow \alpha \notin B$, 即 α 非极小元, 必存在 $\beta' \in B(\beta' \prec \alpha)$, 矛盾. 因此 $\forall \alpha \in A, \exists \beta \in B(x^\beta | x^\alpha)$, 即 $x^\alpha \in \langle x^B \rangle \Rightarrow \langle x^A \rangle \subset \langle x^B \rangle$. 又 $B \subset A \Rightarrow \langle x^B \rangle \subset \langle x^A \rangle$, 因而 $\langle x^A \rangle = \langle x^B \rangle$.

下面我们证明 B 是有限集. 对于 $n = 1$ 的情况, 由于 $\preccurlyeq$ 是全序, 则 B 是单元集, 命题显然成立. 下面假设命题对于 $n - 1$ 的情况也是成立的, 命

$$A^* = \{(\alpha_1, \alpha_2, \ldots, \alpha_{n-1} \in \mathbb{N}^{n-1} | \exists \alpha_n \in \mathbb{N}, (\alpha_1, \ldots, \alpha_n) \in A\},$$

则 A^* 的极小元集 B^* 是有限集. $\forall \beta^* = (\beta_1, \ldots, \beta_{n-1}) \in B^*$, 我们可取 $b_{\beta^*} \in \mathbb{N}$ 使得 $(\beta^*, b_{\beta^*}) \in A$, 由于 B^* 的有限性, 我们可取最大值

$$b = \max\{b_{\beta^*} | \beta^* \in B^*\}.$$

于是 $\forall \alpha \in A$, $\exists \beta^* \in B^*$ 使得 $\beta^* \preccurlyeq (\alpha_1, \ldots, \alpha_{n-1})$, 假设 $\alpha_n > b$, 则

$$(\beta^*, b_{\beta^*}) \preccurlyeq (\beta^*, b) < \alpha,$$

即 α 不是极小元. 因此 A 中任一极小元 α 必满足 $\alpha_n \leq b$, 那么 $\#B \leq (\#B^*) \times (b + 1)$ 是有限的. 由归纳法, 本定理得证. □

引理13.3. *I 是 R 中任一理想, 若 $G \subset I$ 是有限集且 $\langle \mathrm{lt}(G) \rangle = \langle \mathrm{lt}(I) \rangle$, 则 $\langle G \rangle = I$.*

证明. 设 $G = \{g_1, \ldots, g_t\}$, 则 $\forall f \in I$, 由带余除法可得到

$$f = q_1 g_1 + \cdots + q_t g_t + r,$$

其中 r 不可再被 G 约化. 而由于 $r = f - q_1 g_1 - \cdots - q_t g_t \in I$, 于是 $\mathrm{lt}(r) \in \mathrm{lt}(I) \Rightarrow \mathrm{lt}(r) \in \langle \mathrm{lt}(G) \rangle$, 即 r 中每个单项都在 $\langle \mathrm{lt}(G) \rangle$ 中, 因此 $r = 0$, 则 $f \in \langle G \rangle \Rightarrow \langle G \rangle = I$. □

由于任何单项理想均可有限生成, 我们有下面的:

定理13.8 (Hilbert 基定理)**.** *R 中任何理想 I 均可有限生成.*

推论13.3 (理想升链定理)**.** *设 R 中有一理想升链 $I_1 \subset I_2 \subset \cdots \subset I_n \subset \cdots$, 则存在 $n \in \mathbb{N}(\forall m > n, I_m = I_n)$.*

这可由 $I = \bigcup_{i=1}^{\infty} I_i$ 是有限生成的得到. 满足这样条件的环也叫 Noether 环(Noetherian Domain).

13.3.3 Gröbner 基及其性质

现在引入 Gröbner 基的定义:

定义13.16. 设有多项式环 R 中的理想 I 和某一单项序 $<$, I 的有限子集 G 当满足 $\langle \mathrm{lt}(G)\rangle = \langle \mathrm{lt}(I)\rangle$ 时, 称为 I 的 Grönber 基.

我们记理想 I 的全体 Gröbner 基为 $GB(I)$, 即

$$GB(I) = \{G \in 2^I | G是I的\text{ Grönber 基}\}.$$

Gröbner 基的存在性是由 Hilbert 基定理和引理 13.3 保证的, 它有如下的性质:

定理13.9. *设 $G \in GB(I)$, $\forall f \in R$, 存在唯一的 $r \in R$ 使得 $f - r \in I$ 且 r 中无单项可被 $\mathrm{lt}(G)$ 中元素整除, 即不可被 G 约化.*

当考察的 G 是 Gröbner 基时, 我们也记 $f \operatorname{rem} G = \overline{f}^G$ 或 $\overline{f}$.

证明. 存在性由带余除法算法得到, 对于唯一性, 可令 $f = h_1 + r_1 = h_2 + r_2$, 其中 r_1, r_2 分别是两种不同途径约化的结果. 则 $r_1 - r_2 = h_2 - h_1 \in I \Rightarrow \mathrm{lt}(r_1 - r_2)$ 可被 $\mathrm{lt}(G)$ 中元素整除. 由 r_1, r_2 的定义可知 $r_1 - r_2 = 0$. □

推论13.4. $f \in I \Leftrightarrow r = \overline{f} = f \operatorname{rem} G = 0$.

至此, 我们得到了 Gröbner 基的一个优美的性质.

构造理想的 Gröbner 基需要所谓的 S-多项式, 下面简要讨论之.

定义13.17. 对于非零多项式 g, h, 设 $\alpha = \deg g$, $\beta = \deg h$, $x^\gamma = \mathrm{lcm}(x^\alpha, x^\beta)$, 即 $\gamma = (\max(\alpha_1, \beta_1), \ldots, \max(\alpha_n, \beta_n))$, 则定义 g, h 的 S-多项式为

$$S(g,h) = \left(\frac{x^\gamma}{\mathrm{lt}(g)}g - \frac{x^\gamma}{\mathrm{lt}(h)}h\right) \in R.$$

引理13.4. *设 $g_1, \ldots, g_s \in R$, $\alpha_1, \ldots, \alpha_s \in \mathbb{N}^n$, $c_1, \ldots, c_s \in F \setminus \{0\}$,*

$$f = \sum_{1 \le i \le s} c_i x^{\alpha_i} g_i \in R,$$

且有 $\delta \in \mathbb{N}^n$ 使 $\alpha_i + \deg g_i = \delta (1 \le i \le s)$, $\deg f < \delta$, 即这些多项式求和后领项消去 (Leading term cancellation).

令 $x^{\gamma_{ij}} = \mathrm{lcm}(\mathrm{lm}(g_i), \mathrm{lm}(g_j))$, 则存在 $c_{ij} \in F$ 使得 $x^{\gamma_{ij}} | x^\delta$ 且

$$f = \sum_{1 \le i < j \le s} c_{ij} x^{\delta - \gamma_{ij}} S(g_i, g_j),$$

且 $\deg x^{\delta-\gamma_{ij}} S(g_i, g_j) < \delta, (1 \le i < j \le s)$.

证明. 可假定 $\mathrm{lc}(g_i)=1$, 否则可将其归并入 c_i 中而使定理条件形式仍不变, 于是 $\mathrm{lt}(g_i)=\mathrm{lm}(g_i)=x^{\deg g_i}$. 由于 $x^\delta=x^{\alpha_i}\,\mathrm{lm}(g_i)=x^{\alpha_j}\,\mathrm{lm}(g_j)$, 则有 $x^{\gamma_{ij}}|x^\delta$.

由于

$$S(g_i,g_j)=\frac{x^{\gamma_{ij}}}{\mathrm{lt}(g_i)}g_i-\frac{x^{\gamma_{ij}}}{\mathrm{lt}(g_j)}g_j,$$

首项消去告诉我们 $\deg S(g_i,g_j)<\gamma_{ij}$, 因此 $\deg x^{\delta-\gamma_{ij}}S(g_i,g_j)\prec\delta$.

不妨设 $s\geq 2$, 则

$$\begin{aligned}
g&=f-c_1x^{\delta-\gamma_{ij}}S(g_1,g_2)\\
&=c_1x^{\alpha_1}g_1+c_2x^{\alpha_2}g_2+\sum_{3\leq i\leq s}c_ix^{\alpha_i}g_i-c_1x^{\delta-\gamma_{12}}\left(\frac{x^{\gamma_{12}}}{\mathrm{lt}(g_1)}g_1-\frac{x^{\gamma_{12}}}{\mathrm{lt}(g_2)}g_2\right)\\
&=c_1(x^{\alpha_1}-x^{\delta-\deg g_1})g_1+(c_2x^{\alpha_2}+c_1x^{\delta-\deg g_2})g_2+\sum_{3\leq i\leq s}c_ix^{\alpha_i}g_i\\
&=(c_1+c_2)x^{\alpha_2}g_2+\sum_{3\leq i\leq s}c_ix^{\alpha_i}g_i,
\end{aligned}$$

显然 $\deg g<\delta$, 由此时 g 的形式和归纳法, 我们可以证明 f 可以表成定理中的形式. □

下面的定理给出了判别 Gröbner 基的一个充要条件:

定理13.10. $G=\{g_1,\dots,g_s\}\in GB(I)\Leftrightarrow\forall(1\leq i<j\leq s),S(g_i,g_j)\operatorname{rem}G=0.$

证明. $\Rightarrow$. $S(g_i,g_j)\in I=\langle G\rangle\Rightarrow\overline{S(g_i,g_j)}=0$.

$\Leftarrow$. 令 $f\in I\setminus\{0\}$, 由定义只需证明 $\mathrm{lt}(f)\in\langle\mathrm{lt}(G)\rangle$ 即可.

不妨设 $f=\sum_{1\leq i\leq s}q_ig_i$, $\delta=\max\{\deg(q_ig_i)|1\leq i\leq s\}$, 则显然 $\deg f\leq\delta$. 倘若等号不成立, 即 $\deg f<\delta$, 定义

$$f^*=\sum_{1\leq i\leq s,\deg(q_ig_i)=\delta}\mathrm{lt}(q_i)g_i,$$

它显然满足引理 13.4 的条件, 可写为 $S(g_i,g_j)$ 的线性组合, 因而其在 G 下约化为零. 由带余除法, 存在 q_i^* 使得 $f^*=\sum_{1\leq i\leq s}q_i^*g_i$ 且 $\max\{\deg(q_i^*g_i)|1\leq i\leq s\}\leq\deg f^*<\delta$. 由 f^* 的定义可知 $f-f^*=\sum_{1\leq i\leq s}q_i^{**}g_i$, $\max\{\deg(q_i^{**}g_i)|1\leq i\leq s\}<\delta$. 于是 f 也可以表示成

$$f=\sum_{1\leq i\leq s}q_ig_i,\quad\max\{\deg(q_ig_i)|1\leq i\leq s\}<\delta,$$

这与 δ 的定义矛盾, 故 $\deg f=\delta$, 即 $\exists i$ 使 $\deg f=\deg(q_ig_i)$, 因此

$$\mathrm{lt}(f)=\sum_{1\leq i\leq s,\deg(q_ig_i)=\delta}\mathrm{lt}(q_i)\,\mathrm{lt}(g_i)\in\langle\mathrm{lt}(G)\rangle.$$

证毕. □

13.3.4 Buchberger 算法及约化 Gröbner 基

Buchberger 提出了 Gröbner 基的概念并给出了计算它的方法, 即下面的 Buchberger 算法:

算法13.3 (Buchberger 算法).
输入:$f_1, \ldots, f_s \in R$,
输出:$I = \langle f_1, \ldots, f_s \rangle$ 的一个 Grönber 基 G.

1. $G = \{f_1, \ldots, f_s\}$,
2. 循环作后面所有步骤,
3. $S = \varnothing$, 将 G 中元素编号为 $G = \{g_1, \ldots, g_t\}$,
4. 对于所有的序对 $(i, j), 1 \leq i < j \leq t$ 计算 $r = S(g_i, g_j) \operatorname{rem} G$, 若 $r \neq 0$ 则 $S = S \bigcup \{r\}$,
5. 若 $S = \varnothing$ 则输出 G, 否则 $G = G \bigcup S$.

算法有效性. 我们只需证明该算法循环是可以终止的, 因为终止时由定理 13.10 可知 G 即是 Gröbner 基. 因为每步循环之后我们都可以得到一个新的集合 G, 我们给它们编上号, 记为 $G_1 \subset G_2 \subset \cdots$, 显然

$$\langle \operatorname{lt}(G_1) \rangle \subset \langle \operatorname{lt}(G_2) \rangle \subset \cdots,$$

由理想升链定理, $\exists n \in \mathbb{N}$ 使得 $\forall m > n$ 有 $\langle \operatorname{lt}(G_n) \rangle = \langle \operatorname{lt}(G_m) \rangle$, 此时 $\forall g, h \in G_n$, 令 $r = S(g, h) \operatorname{rem} G_n$, 则显然 $r = 0 \vee r \in G_{n+1}$, 于是 $\operatorname{lt}(r) \in \langle \operatorname{lt}(G_{n+1}) \rangle = \langle \operatorname{lt}(G_n) \rangle$, 由 r 是多项式对 G_n 的约化结果知道 $r = 0$, 于是算法终止. □

多项式理想的 Gröbner 基并不是唯一的, 而且在上面的算法中 G 的规模的增长是十分迅速的, 求出的结果中可能含有大量的多项式, 因此我们要对 Gröbner 基做一定的优化, 下面我们一步一步对其进行约化化简.

引理13.5. 设 $G \in GB(I)$, $g \in G$ 且 $\operatorname{lt}(g) \in \langle \operatorname{lt}(G \setminus \{g\}) \rangle$, 则 $G \setminus \{g\} \in GB(I)$.

证明. $\operatorname{lt}(g) \in \langle \operatorname{lt}(G \setminus \{g\}) \rangle \Rightarrow \langle \operatorname{lt}(G \setminus \{g\}) \rangle = \langle \operatorname{lt}(G) \rangle = \langle \operatorname{lt}(I) \rangle \Rightarrow G \setminus \{g\} \in GB(I)$. □

定义13.18. 设 $G \in GB(I)$, 且 $\forall g \in G$, 有 $\mathrm{lc}(g) = 1 \wedge \mathrm{lt}(g) \notin \langle \mathrm{lt}(G \setminus \{g\}) \rangle$, 则称 G 是 I 的极小 Gröbner 基(Minimal Gröbner basis), 并简记为 $G \in MGB(I)$.

定义13.19. 设 $G \in MGB(I)$, 若对于 $g \in G$, g 不可再被 $G \setminus \{g\}$ 约化, 即 g 中任何一单项均不在理想 $\langle \mathrm{lt}(G \setminus \{g\}) \rangle$ 中, 则称 g 关于 G 是约化的. 若 $\forall g \in G$, g 关于 G 都是约化的, 则称 G 是约化 Gröbner 基(Reduced Gröbner basis), 并简记为 $G \in RGB(I)$(或由下面的唯一性可记为 $G = RGB(I)$).

定理13.11. *每个多项式理想 I 都有唯一的约化 Gröbner 基.*

证明. 存在性. 由引理 13.5 可将 G 化为 I 的极小 Gröbner 基, 设此时 $G = \{g_1, g_2, \ldots, g_s\}$, 然后对 $1 \leq i \leq s$ 归纳地做 $h_i = g_i \operatorname{rem}\{h_1, \ldots, h_{i-1}, g_{i+1}, \ldots, g_s\}$. 由极小 Gröbner 基的条件知道 $\mathrm{lt}(h_i) = \mathrm{lt}(g_i), (1 \leq i \leq s)$. 于是由 h_i 相对于 $\{h_1, \ldots, h_{i-1}, g_{i+1}, \ldots, g_s\}$ 约化可知其相对于 $G_s = \{h_1, \ldots, h_s\}$ 也是约化的.

唯一性. 设 G, G^* 均是 I 的约化 Gröbner 基, 则 $\forall g \in \mathrm{lt}(G) \subset \langle \mathrm{lt}(G) \rangle = \langle \mathrm{lt}(G^*) \rangle$, $\exists g^* \in G^*$ 使 $\mathrm{lt}(g^*) | \mathrm{lt}(g)$, 同样地, $\exists g^{**} \in G$ 使 $\mathrm{lt}(g^{**}) | \mathrm{lt}(g^*)$, 因此 $\mathrm{lt}(g^{**}) | \mathrm{lt}(g)$, 由于约化 Gröbner 基也是极小 Gröbner 基, 我们知道 $\mathrm{lt}(g) = \mathrm{lt}(g^{**}) = \mathrm{lt}(g^*) \in \mathrm{lt}(G^*) \Rightarrow \mathrm{lt}(G) \subset \mathrm{lt}(G^*)$, 再由对称可证 $\mathrm{lt}(G) = \mathrm{lt}(G^*)$.

$\forall g \in G$, 取 $g^* \in G^*$ 使得 $\mathrm{lt}(g) = \mathrm{lt}(g^*)$. 由于 G, G^* 约化, 则 $g - g^* \in I$ 中任一单项式均不能被 $\mathrm{lt}(G \setminus \{g\}) = \mathrm{lt}(G^* \setminus \{g^*\})$ 中元素约化. 于是 $g - g^* = g - g^* \operatorname{rem} G = 0 \Rightarrow g = g^* \in G^* \Rightarrow G \subset G^* \Rightarrow G = G^*$. □

引理 13.5 给出了由 Gröbner 基求极小 Gröbner 基的方法, 定理 13.11 的存在性证明中也给出了极小 Gröbner 基构造约化 Gröbner 基的方法.

13.3.5 Buchberger 算法的两个改进

Buchberger 算法计算过程中集合 G 中的多项式会越来越多, 呈指数规模增长, 因而需要对其作一定的优化. [17]3.3 节提出了两种改进的方法. 首先我们将算法 13.3 重新描述如下, 以便于我们后面叙述改进算法.

算法13.4 (Buchberger 算法).

输入输出同算法 13.3 ,

1. $G = \{f_1, \ldots, f_s\}$, $H = \{\{i, j\} | i \neq j \wedge 1 \leq i, j \leq s\}$,
2. 当 $H \neq \varnothing$ 时循环做后面两步,
3. 任取 $h = \{i, j\} \in H$, $H = H \setminus \{h\}$, $r = S(f_i, f_j) \operatorname{rem} G$,

4. 若 $r \neq 0$ 则 $f_{s+1} = r$, $H = H \bigcup \{\{i, s+1\} | 1 \leq i \leq s\}$, $G = G \bigcup \{f_{s+1}\}$, $s = s+1$,

5. 输出 G.

注175. 设 $E = \{f_1, \ldots, f_s\}$, 如果我们要得到矩阵 M 使得 $G = EM$, 那么首先令 $M_s = I_{s\times s}$, 然后在上面算法每次第 4 步判断成功后, 设带余除法给出 $r = S(f_i, f_j) \operatorname{rem} G = S(f_i, f_j) - q_1 f_1 - \cdots - q_s f_s$, 设 $S(f_i, f_j) = \sum_{1\leq k\leq s} S_k f_k$, 其中 $S_i = x^{\gamma_{ij}} / \operatorname{lt}(g_i)$, $S_j = -x^{\gamma_{ij}} / \operatorname{lt}(g_j)$, 其余的 $S_k = 0$. 则由

$$(f_1, \ldots, f_s, r) = (f_1, \ldots, f_s) \begin{pmatrix} 1 & & & S_1 - q_1 \\ & 1 & & S_2 - q_2 \\ & & \ddots & \vdots \\ & & 1 & S_s - q_s \end{pmatrix} =: (f_1, \ldots, f_s) Q_s,$$

可知有迭代 $M_{s+1} = M_s Q_s$, 输出最后的 $M = M_s$ 即可.

注176. 产生极小 Gröbner 基时只要从 M 中去掉相应的列, 而对于约化过程, 同样由带余除法得到 $q_1, \ldots, q_s$, 乘以相应的迭代矩阵.

第一个改进

引理13.6. *设有多项式 $f_1, \ldots, f_s$ 和 d, $I = \langle f_1, \ldots, f_s \rangle$, $J = \langle f_1 d, \ldots, f_s d \rangle$, 则 $\{f_1, \ldots, f_s\} \in GB(I) \Leftrightarrow \{f_1 d, \ldots, f_s d\} \in GB(J)$.*

此引理由 Gröbner 基的定义 $\langle \operatorname{lt}(G) \rangle = \langle \operatorname{lt}(I) \rangle$ 可证.

引理13.7. *设有非零多项式 f, g, $I = \langle f, g \rangle$, $d = \gcd(f, g)$, 则下面两个条件等价:*

(1) $\operatorname{lm}(f/d)$ 与 $\operatorname{lm}(g/d)$ 互素,

(2) $S(f, g) \operatorname{rem} \{f, g\} = 0$, 即 $\{f, g\} \in GB(I)$.

证明. (1)⇒(2). 先设 $d = \gcd(f, g) = 1$, 且 $f = aX + f'$, $g = bY + g'$, 其中 $\operatorname{lc}(f) = a, \operatorname{lm}(f) = X, \operatorname{lc} g = b, \operatorname{lm}(g) = Y$, f', g' 是余下的部分, 于是

$$X = \frac{f - f'}{a}, \quad Y = \frac{g - g'}{b}.$$

(I)若 $f' = g' = 0$, 则 $S(f, g) = 0$,

(II)若 $f' = 0, g' \neq 0$, 由 $\gcd(\operatorname{lm}(f), \operatorname{lm}(g)) = 1$ 得

$$S(f, g) = \frac{1}{a} Y f - \frac{1}{b} X g = \frac{1}{ab}(g - g') f - \frac{1}{ab} f g = -\frac{1}{ab} g' f,$$

若它能被 g 约化, 则由 $\mathrm{lm}(g)|\mathrm{lm}(g'f) \wedge \gcd(\mathrm{lm}(g),\mathrm{lm}(f)) = 1$ 知 $\mathrm{lm}(g)|\mathrm{lm}(g')$, 这与 $\deg g' < \deg g \wedge g' \neq 0$ 矛盾.

(III)若 $f' \neq 0, g' = 0$, 这与情形(II)类似.

(IV)若 $f' \neq 0 \wedge g' \neq 0$, 此时经过计算可知

$$S(f,g) = \frac{1}{ab}(f'g - g'f),$$

我们断言 $\mathrm{lm}(f'g) \neq \mathrm{lm}(g'f)$. 假设二者相等, 则 $\mathrm{lm}(f')\mathrm{lm}(g) = \mathrm{lm}(g')\mathrm{lm}(f)$, 由于 $\mathrm{lm}(f)$ 和 $\mathrm{lm}(g)$ 互素, 我们得到 $\mathrm{lm}(f)|\mathrm{lm}(f')$, 矛盾. 不妨设此时 $\mathrm{lm}(f'g) > \mathrm{lm}(g'f)$, 则第一步仅可通过 g 约化, 其结果为

$$S(f,g) \to \frac{1}{ab}[(f' - \mathrm{lt}(f'))g - g'f],$$

对于相反的情形如法炮制, 得到第一步的约化结果仍然可进行同样的讨论, 只需将其中的 $f' - \mathrm{lt}(f')$ 看作 f' 即可. 于是这样的约化过程可一直继续, 直至约化为 0.

综上, 我们在 $d = 1$ 的情况下证明了(1)⇒(2). 当 $d \neq 1$ 时, 我们有 $\gcd(f/d, g/d) = 1$, 于是 $S(f/d, g/d)\,\mathrm{rem}\{f/d, g/d\} = 0$, 即 $\{f/d, g/d\}$ 是 Gröbner 基, 由引理 13.6 可知 $\{f, g\}$ 也是 Gröbner 基, 命题得证.

(2)⇒(1). (I)首先我们仍然设 $d = \gcd(f,g) = 1$, 取单项式 $D, X, Y \in M$ 满足

$$\mathrm{lm}(f) = DX, \quad \mathrm{lm}(g) = DY, \quad \gcd(X,Y) = 1,$$

于是 $S(f,g) = \frac{Y}{\mathrm{lc}(f)}f - \frac{X}{\mathrm{lc}(g)}g$, 由假设 $\{f,g\}$ 是 Gröbner 基, 我们进行带余除法可以得到多项式 u, v 使得 $S(f,g) = uf + vg$, 且 $\mathrm{lm}(uf), \mathrm{lm}(vg) \leq \mathrm{lm}(S(f,g))$. 整理可得

$$\left(\frac{X}{\mathrm{lc}(g)} + v\right)g = \left(\frac{Y}{\mathrm{lc}(f)} - u\right)f,$$

因此 $g|(Y/\mathrm{lc}(f) - u)$, 又

$$\mathrm{lm}(u)DX = \mathrm{lm}(uf) \leq \mathrm{lm}(S(f,g)) < \mathrm{lm}(Yf) = \mathrm{lm}(Xg) = DXY \Rightarrow \mathrm{lm}(u) < Y,$$

则 $g|(Y/\mathrm{lc}(f) - u) \Rightarrow DY|Y \Rightarrow D = 1$, 亦即 $\mathrm{lm}(f)$, $\mathrm{lm}(g)$ 互素.

(II)当 $d \neq 1$ 时, 则 $\gcd(f/d, g/d) = 1$, 于是由 $\{f,g\}$ 是 Gröbner 基可知 $\{f/d, g/d\}$ 是 Gröbner 基, 因而 $\mathrm{lm}(f/d)$, $\mathrm{lm}(g/d)$ 互素. □

定理13.12. *$S(f,g)\,\mathrm{rem}\{f,g\} = 0$ 的一个充分条件是 $\gcd(\mathrm{lm}(f),\mathrm{lm}(g)) = 1$.*

注177. 只需注意到 $\gcd(\mathrm{lm}(f),\mathrm{lm}(g)) = 1 \Rightarrow \gcd(f,g) = 1$.

我们可以由此得到 Buchberger 算法的第一个修正, 在计算 S-多项式并约化之前由 $\mathrm{lm}(f), \mathrm{lm}(g)$ 是否互素来决定是否要计算.

第二个改进

定义13.20. 对于多项式 $f_1,\dots,f_s$, 设 $f=(f_1,\dots,f_s)\in R^s$, 定义 $\mathrm{Syz}(E)=\{h=(h_1,\dots,h_s)\in R^s|h\cdot f=0\}$.

定理13.13. 设 $c_1,\dots,c_s\in F\setminus\{0\}$, $X_1,\dots,X_s\in M$, $X_{ij}=\mathrm{lcm}(X_i,X_j)$, 则 $\mathrm{Syz}(c_1X_1,\dots,c_sX_s)$ 由

$$\left\{\tau_{ij}=\frac{X_{ij}}{c_iX_i}e_i-\frac{X_{ij}}{c_jX_j}e_j\in R^s|1\le i<j\le s\right\}$$

生成, 其中 e_i,e_j 为 R^s 中的自然基矢.

证明. 首先易验证 $\langle\tau_{ij}\rangle\subset\mathrm{Syz}(c_1X_1,\dots,c_sX_s)$. 现在假设 $h=(h_1,\dots,h_s)\in\mathrm{Syz}(c_1X_1,\dots,c_sX_s)$, 于是有

$$h_1c_1X_1+\cdots+h_sc_sX_s=0.$$

考虑任一单项式 $X\in M$, 则在上式中含 X 的同类项合并之后为 0, 因此我们可只考虑这些项, 将其分离出来. 可设 $h_i=c_i'X_i'$, 其中 $c_i'=0$ 或 $X_i'X_i=X$, 设 $c_{i_1}',\dots,c_{i_t}'$ 是 c_i' 中不为零的系数重新编号, 于是有 $c_1'c_1+\cdots+c_s'c_s=c_{i_1}'c_{i_1}+\cdots+c_{i_t}'c_{i_t}=0$, 则

$$\begin{aligned}
h=&(h_1,\dots,h_s)=c_{i_1}'X_{i_1}'e_{i_1}+\cdots+c_{i_t}'X_{i_t}'e_{i_t}\\
&=c_{i_1}'c_{i_1}\frac{X}{c_{i_1}X_{i_1}}e_{i_1}+\cdots+c_{i_t}'c_{i_t}\frac{X}{c_{i_t}X_{i_t}}e_{i_t}\\
&=c_{i_1}'c_{i_1}\frac{X}{X_{i_1i_2}}(\frac{X_{i_1i_2}}{c_{i_1}X_{i_1}}e_{i_1}-\frac{X_{i_1i_2}}{c_{i_2}X_{i_2}}e_{i_2})\\
&+(c_{i_1}'c_{i_1}+c_{i_2}'c_{i_2})\frac{X}{X_{i_2i_3}}(\frac{X_{i_2i_3}}{c_{i_2}X_{i_2}}e_{i_2}-\frac{X_{i_2i_3}}{C_{i_3}X_{i_3}}e_{i_3})+\cdots\\
&+(c_{i_1}'c_{i_1}+\cdots+c_{i_{t-1}}'c_{i_{t-1}})\frac{X}{X_{i_{t-1}i_t}}(\frac{X_{i_{t-1}i_t}}{c_{i_{t-1}}X_{i_{t-1}}}e_{i_{t-1}}-\frac{X_{i_{t-1}i_t}}{c_{i_t}X_{i_t}}e_{i_t})\\
&+(c_{i_1}'c_{i_1}+\cdots+c_{i_t}'c_{i_t})\frac{X}{c_{i_t}X_{i_t}}e_{i_t}.
\end{aligned}$$

注意到上式最后一项为零, 得证. □

定理13.14. 设 $G=\{g_1,\dots,g_s\}$, $B\{\tau_{ij}|1\le i<j\le s\}$ 是 $\mathrm{Syz}(\mathrm{lt}(g_1),\dots,\mathrm{lt}(g_s))$ 的生成元集, 则 $G\in GB(\langle G\rangle)\Leftrightarrow\forall h=(h_1,\dots,h_s)\in B(h_1g_1+\cdots+h_sg_s\,\mathrm{rem}\,G=0)$.

证明. 注意到 $\tau_{ij}\cdot(g_1,\dots,g_s)=S(g_i,g_j)$, 则命题显然. □

引理13.8. *记* $X_{ij}=\mathrm{lcm}(X_i,X_j)$, $X_{ijl}=\mathrm{lcm}(X_i,X_j,X_l)$, *则*

$$\frac{X_{ijl}}{X_{ij}}\tau_{ij}+\frac{X_{ijl}}{X_{jl}}\tau_{jl}+\frac{X_{ijl}}{X_{li}}\tau_{li}=0,$$

若 $X_l|X_{ij}$, *则* τ_{ij} *在* τ_{jl} *和* t_{li} *生成的* R^s *的子模中.*

注178. 等式可以由 τ_{ij} 的定义式直接验证, 对于第二个断言利用 $X_l|X_{ij}$ 时 $X_{ijl}=X_{ij}$ 代入等式直接得.

推论13.5. *设* $B\subset\{\tau_{ij}|1\leq i<j\leq s\}$ *是* $\mathrm{Syz}(c_1X_1,\ldots,x_sX_s)$ *的生成元集, 若* $\exists i,j,l$ *使* $\tau_{ij},\tau_{jl},\tau_{li}\in B$, *且* $X_l|X_{ij}$, *则* $B\setminus\{\tau_{ij}\}$ *也是* $\mathrm{Syz}(c_1X_1,\ldots,c_sX_s)$ *的生成元集.*

由此推论我们可以得到 Buchberger 算法的第二个改进, 即某些 S-多项式可能是其它两个 S-多项式的线性组合, 可以不予计算. 这一步, 能够显著提高 Buchberger 算法的效率.

改进后的算法

鉴于前文的分析, 我们现在可以给出 Buchberger 算法的改进算法.

算法13.5 (改进 Buchberger 算法)**.**

输入:多项式 $f_1,\ldots,f_s$,

输出:理想 $\langle f_1,\ldots,f_s\rangle$ 的 Gröbner 基 G.

1. $G=\{f_1,\ldots,f_s\}$, $C=\varnothing$, $NC=\{\{1,2\}\}$, $i=2$,
2. 当 $i<s$ 时, 循环做:$NC=NC\bigcup\{\{j,i+1\}|1\leq j\leq i\}$, $NC=\mathrm{crit}(NC,C,i+1)$, $i=i+1$,
3. 当 $NC\neq\varnothing$ 时, 循环做后面所有步骤, 否则输出 G 退出,
4. 任选 $\{i,j\}\in NC$, 令 $NC=NC\setminus\{\{i,j\}\}$, $C=C\bigcup\{\{i,j\}\}$,
5. 若 $\gcd(\mathrm{lm}(f_i),\mathrm{lm}(f_j))\neq 1$, 则做下面 6, 7 步,
6. $r=S(f_i,f_j)\,\mathrm{rem}\,G$,
7. 若 $r\neq 0$ 则 $f_{s+1}=r, G=G\bigcup\{f_{s+1}\}$, $s=s+1$, $NC=NC\bigcup\{\{i,s\}|1\leq i\leq s-1\}$, $NC=\mathrm{crit}(NC,C,s)$.

上面算法中 crit 函数是第二个改进判别法, 由下面算法给出, 其中的 $X_i = \mathrm{lm}(f_i)$:

算法13.6 (crit 函数).

$\mathrm{crit}(NC, C, s)$, 输出简化后的 NC.

1. $l = 1$,
2. 当 $l < s$ 时循环做下面 3–7 步, 否则转 8 步,
3. 若 $\{l, s\} \in NC$ 则令 $i = 1$ 并做下面 4–6 步, 否则转 7 步,
4. 当 $i < s$ 时循环做下面 5, 6 步, 否则转 7 步,
5. 若 $\{i, l\} \in NC \bigcup \wedge \{i, s\} \in NC$ 则看 $X_l | \mathrm{lcm}(X_i, X_s)$ 是否成立, 是则令 $NC = NC \setminus \{\{i, s\}\}$,
6. $i = i + 1$,
7. $l = l + 1$,
8. $i = 1$,
9. 若 $i < s$ 则做下面 10–14 步, 否则转 15 步,
10. 若 $\{i, s\} \in NC$ 则令 $j = i + 1$ 并做下面 11–13 步, 否则转 14 步,
11. 当 $j < s$ 时做下面 12, 13 步, 否则转 14 步,
12. 若 $\{j, s\} \in NC \wedge \{i, j\} \in NC$ 则看 $X_s | \mathrm{lcm}(X_i, X_j)$ 是否成立, 是则令 $NC = NC \setminus \{\{i, j\}\}$,
13. $j = j + 1$,
14. $i = i + 1$,
15. 输出 NC.

13.3.6 Gröbner 基的应用

一些简单的应用

现在我们可以回到本章开头所提出的一些关于多元多项式理想的问题上来了. 我们要解决的第一个问题是对于任何一个多项式 f, 如何判断它在不在一个已知的理想 $I=\langle f_1,\dots,f_s\rangle$ 中, 以及找出多项式 $v_1,\dots,v_s$ 使得 $f=v_1f_1+\cdots+v_sf_s$.

首先我们根据前面生成 Gröbner 基的算法, 不仅得到了约化的 Gröbner 基 $G=\{g_1,\dots,g_t\}$, 而且可以得到变换矩阵 $M_{t\times s}$ 使得 $(g_1,\dots,g_t)=(f_1,\dots,f_s)M$. 由推论 13.4 利用带余除法可判定 f 是不是在理想 I 中. 若 $f\in I$, 设利用除法算法得到的多项式为 $u_1,\dots,u_t$, 满足 $f=u_1g_1+\cdots+u_tg_t$, 于是由

$$f=(g_1,\dots,g_t)\begin{pmatrix}u_1\\ \vdots\\ u_t\end{pmatrix}=(f_1,\dots,f_s)M\begin{pmatrix}u_1\\ \vdots\\ u_t\end{pmatrix}$$

知 $(v_1,\dots,v_s)^T=M(u_1,\dots,u_t)^T$.

对于两个多项式理想是否相等的判定, 也可由它们唯一的约化 Gröbner 基来进行. 而在商环 R/I 中的算术需要用到代表元, 我们也可取为 $\overline{f}=f\operatorname{rem}G$. 下面定理给出了商环 R/I 的一组基.

定理13.15. $\mathcal{B}=\{\overline{g}|g\in M\wedge g\notin\langle\mathrm{lt}(G)\rangle\}$ 是 R/I 在 F 上的一组基.

R/I 中的求逆问题. 设 $f\in R/I$, 我们既已知道了该环的基, 则可设 f^{-1} 为 $\mathcal{B}$ 的元素的线性组合来求解, 这比较复杂. 我们设 g 是 f 的逆, 注意到

$$fg-1\in I\Leftrightarrow 1\in\langle I,f\rangle\Leftrightarrow\langle I,f\rangle=R,$$

则我们可以求 $\langle I,f\rangle$ 的 Gröbner 基, 看 1 是否在其中来确定是否存在逆. 再求出 1 在 $\langle I,f\rangle$ 中的线性表示, 即 $1=v_1f_1+\cdots+v_sf_s+gf$, g 即是 f 在 R/I 中的逆.

Hilbert 零点定理及 Gröbner 基在解方程中的应用

现在考虑 F 的某个扩域 $k\supset F$(或$k=F$).

定义13.21. $V_k(I)=\{a\in k^n|\forall f\in I(f(a)=0)\}$.

对于 F^n 中子集 V, 定义 R 的理想 $I(V)=\{f\in R|f(V)=0\}$.

定理13.16 (弱 Hilbert 零点定理)**.** $I\subset R$, k 是 F 的代数闭域, 则 $V_k(I)=\varnothing\Leftrightarrow I=R=F[x]$.

定义13.22. 定义多项式理想 I 的根理想(radical)为 $\sqrt{I}=\{f\in R|\exists e\in\mathbb{N}(f^e\in I)\}$.

显然有 $\forall k\supset F$, $V_k(I)=V_k(\sqrt{I})$.

定理13.17 (强 Hilbert 零点定理)**.** $I(V_k(I))=\sqrt{I}$.

推论13.6. $V_k(I)=V_k(J)\Leftrightarrow\sqrt{I}=\sqrt{J}$.

注179. 强, 弱 Hilbert 零点定理的证明见有关代数几何的书, 如 [89] 及其中译本 [4].

定理13.18. *设 $G=\{g_1,\dots,g_t\}\in GB(I)$, 下面三个命题等价:*

(1)$V_k(I)$ 是有限集,

(2)$\forall i\in\{1,\dots,n\}$, $\exists v_i\in\mathbb{N}, j\in\{1,\dots,t\}$ 使得 $\mathrm{lm}(g_j)=x_i^{v_i}$,

(3)R/I 有限维.

当上面任何一个条件满足时, 我们也称理想 I 是零维理想(zero-dimensional).

证明. (1)⇒(2). 若 $V_k(I)=\varnothing$ 则 $1\in G$, 取 $v_i=0$ 即可. 下面假设 $V_k(I)\neq\varnothing$, 取某个 $i\in\{1,\dots,n\}$, 对于 $l=\#V_k(I)$, 取 $a_{im}(m=1,2,\dots,l)$ 为 $V_k(I)$ 中点的第 i 个分量, 取非零多项式 $f_m\in F[x_i]\subset F[x]$ 使得 $f_m(a_{im})=0$, 令 $f=f_1\cdots f_l\in F[x_i]\subset F[x]$, 则 $f\in I(V_k(I))=\sqrt{I}$, 于是 $\exists e\in\mathbb{N}$ 使得 $f^e\in I$, 则 $\mathrm{lm}(f^e)$ 是 x_i 的幂, G 中有元素 g_j 满足 $g_j=x_i^{v_i}$ 为 x_i 的幂.

(2)⇒(3). 对于 R/I 的基 $\prod_{1\le i\le n}x_i^{\alpha_i}$, 一定满足 $\alpha_i<v_i$, 维数不超过 $\prod_{1\le i\le n}v_i$, 因此它是有限维的.

(3)⇒(1). 考虑多项式集合 $\{\overline{x_i^j}|j\in\mathbb{N}\}$, 由于在 R/I 中维数有限, 则它们一定线性相关, 即存在 $m\in\mathbb{N}$ 使得 $\sum_{0\le j\le m}c_j\overline{x_i^j}=0$, 亦即 $\sum_{0\le j\le m}c_jx_i^j\in I$, 该多项式在 $F[x_i]$ 中零点有限, 至多为 m 个, 不妨设其零点集为 V_i, 于是 $V_k(I)\subset\prod_{1\le i\le n}V_i$ 是有限集. □

推论13.7. *设 I 是零维理想, $G=\{g_1,\dots,g_t\}=RGB(I)$, 规定的字典序为 $x_1<x_2<\cdots<x_n$, 则 $t\ge n$, 并可将 $g_1,\dots,g_t\in G$ 重新编号使得 $g_i(1\le i\le n)$ 只含 $x_1,x_2,\dots,x_i$ 且 $\mathrm{lm}(g_i)$ 为 x_i 的幂.*

例13.6. 考虑由 $f_1=x^2+y^2+z^2-1, f_2=x+y+z, f_3=x^2-2x+y^2-2y+z^2+2z$ 生成的理想的 Gröbner 基.

解: 由 Buchberger 算法可得其 Gröbner 基为

$$G=\{g_1,g_2,g_3\}=\{16x^2-4x-7,4x+4y-1,4z+1\}.$$

显然, $V(I)$ 是一有限集, 此基的形式正如推论所说, 由此我们可以由第一个一元多项式方程解出其根, 代入第二个方程解出第二个变元, 依次迭代下去即可解出所有的根. 这里消元的结果和例 13.1 利用结式消元得到的结果是一样的. ◇

方程组解的结构的一些讨论

现在为了方便起见, 将所讨论的域限定为复数域 $\mathbb{C}$, 由上一小节所讨论的内容, 我们很容易推广到如下结论:

定理13.19. *I 是零维理想当且仅当 $\forall i \in \{1,\dots,n\}$, 消元理想 $I\bigcap\mathbb{C}[x] \neq \{0\}$.*

证明. ($\Rightarrow$) 显然. 我们只要选取相应的字典序 $x_i < x_1 < \cdots < x_{i-1} < x_{i+1} < \cdots < x_n$ 即可.

($\Leftarrow$). 当消元理想是非平凡理想时, 其是一元多项式环 $\mathbb{C}[x_i]$ 上的理想, 因而是一个主理想, 可设其由 f_i 生成, f_i 首一且 $\deg f_i = m_i$, 则对任何单项序有 $x_i^{m_i} \in \langle \mathrm{lt}(I)\rangle = \langle \mathrm{lt}(G)\rangle$. 由此易得 R/I 是 $\mathbb{C}$ 上有限维线性空间. □

注180. f_i 可由满足 $\sum_{j=0}^{m_i} c_j \overline{x_i}^j = \overline{0}$ 的极小多项式得到.

定义13.23. 记 p_{red} 为多项式 p 的无平方部分, 即 $p_{red} := p/\gcd(p,p')$.

定理13.20. *当 p 是一元多项式时, 有 $\sqrt{\langle p\rangle} = \langle p_{red}\rangle$.*

证明. 首先由 $V(p) = V(p_{red}) \Rightarrow \sqrt{\langle p\rangle} = I(V(p_{red})) = \sqrt{\langle p_{red}\rangle}$. 而显然有 $\sqrt{\langle p_{red}\rangle} = \langle p_{red}\rangle$. □

[9]46 页给出了如下定理:

定理13.21. *设 $I \subset \mathbb{C}[x_1, x_2, \dots, x_n]$ 是一多项式理想, 则有*

$$\sqrt{I} = I + \langle p_{1,red}, p_{2,red}, \dots, p_{n,red}\rangle,$$

其中 p_i 是消元理想 $I\bigcap\mathbb{C}[x_i]$ 的唯一首一生成元, 且 $p_{i,red}$ 是 p_i 的无平方部分.

证明. 设上面等式右边的理想为 J, 先证其为一根理想. 我们先将诸 p_i(记 $n_i = \deg p_i$)进行 $\mathbb{C}$ 上的因子分解, 得到

$$p_{i,red} = \prod_{1\le j\le n_i} (x_i - a_{ij}),$$

其中由无平方部分的性质可知 $a_{i1}, a_{i2}, \dots, a_{in_i}$ 互不相同. 由 $p_{1,red} \in J$ 可得

$$J = J + \langle p_{1,red}\rangle = \bigcap_{1\le j\le n_1} (J + \langle x_1 - a_{1j}\rangle),$$

同样地对于其它 $p_{i,red}$ 可得

$$J = \bigcap_{1\le j_i\le n_i, 1\le i\le n} (J + \langle x_1 - a_{1j_1}, x_2 - a_{2j_2}, \dots, x_n - a_{nj_n}\rangle).$$

对于每一项, $\langle x_1 - a_{1j_1}, \ldots, x_n - a_{nj_n}\rangle$ 都是极大理想, 因而 J 是有限个极大理想的交集, 即有限个根理想的交集, 仍然是根理想.

由 J 的定义显然有 $I \subset J$, 又由于 J 的零点全在 $V(I)$ 中, 于是 $I \subset J \subset \sqrt{I}$, 取根理想有 $\sqrt{I} = \sqrt{J} = J$. □

下面我们给出一个有用的引理:

引理13.9. 设 $S = \{p_1, p_2, \ldots, p_m\} \subset \mathbb{C}^n$ 是 m 个互不相同的点的集合, 则存在多项式组 $g_i \in \mathbb{C}[x_1, \ldots, x_n](1 \le i \le m)$, 使得 $g_i(p_j) = \delta_{ij}$.

证明. 回忆一元多项式情形下的 Lagrange 插值的构造方法, 我们可以类似地构造. 首先考虑 p_1, p_2 两个点, 两个点肯定有某个分量不同, 不妨设 $p_{1,k} \neq p_{2,k}$, 则定义

$$g_{1,2} = \frac{x_k - p_{2,k}}{p_{1,k} - p_{2,k}},$$

其满足 $g_{1,2}(p_1) = 1$, $g_{1,2}(p_2) = 0$, 同样构造出 $g_{1,j}(2 \le j \le m)$ 后, 取 $g_1 = g_{1,2}g_{1,3}\cdots g_{1,m}$ 即可.

对于 $g_i(2 \le i \le m)$ 可类似构造. □

下面的定理给出了方程组根的个数的估计([9]47 页).

定理13.22. I 是 $\mathbb{C}[X]$ 中的零维理想, $A = R/I$, 则 $\dim A \ge \#V(I)$, 取等号当且仅当 $I = \sqrt{I}$.

证明. 设 $V(I) = \{p_1, \ldots, p_m\}$, 其中 $m = \#V(I)$, 考虑线性映射:

$$\delta \subset \mathbb{C}[X]/I \times \mathbb{C}^m : \overline{f} \mapsto (\overline{f}(p_1), \ldots, \overline{f}(p_m)).$$

根据引理 13.9 设 $g_1, \ldots, g_m$ 满足 $g_i(p_j) = \delta_{ij}$, 则 $\overline{g_i}(p_j) = \delta_{ij}$. 由于 $\forall(\lambda_1, \ldots, \lambda_m) \in \mathbb{C}^m$, $\exists f = \sum_{1\le i\le m} \lambda_i g_i$, 使得 $\delta(\overline{f}) = (\lambda_1, \ldots, \lambda_m)$, 因此 δ 是满射. 于是 $\dim A \ge \dim \mathbb{C}^m = m = \#V(I)$.

对于第二个等价条件, 先设 $I = \sqrt{I}$, 令 $\overline{f} \in \ker\delta$, 则 $f(V(I)) = 0 \Rightarrow f \in I(V(I)) = \sqrt{I} = I \Rightarrow \overline{f} = 0$, 从而 δ 是单射, 因而是同构, $\dim A = m$.

若 $\dim A = m$, 首先 $I \subset \sqrt{I}$, $\forall f \in \sqrt{I} = I(V(I))$, 有 $f(V(I)) = 0$, 则 $\delta(\overline{f}) = (0, \ldots, 0) \Rightarrow \overline{f} \in \ker\delta \Rightarrow f \in I \Rightarrow I = \sqrt{I}$. □

13.3.7 Gröbner 基和特征值法解方程组

在用 Gröbner 基理论将方程组化为三角形列后, 采用一步步代入来算会引入累积误差, [9]2.4 节给出了一种方法, 用于求解的任何一个分量, 同时也给出了求消元理想$I \bigcap \mathbb{C}[x_i]$ 的首一生成元的快速方法. 设 $A = R/I$, 首先我们定义

定义13.24. 线性映射 $m_f \subset A \times A$ 使得 $m_f(\overline{g}) = \overline{f}\overline{g}$, 显然 $m_f = 0 \Leftrightarrow \overline{f} = 0$. 以后 m_f 既指线性映射, 也可指其在 A 的某组基上的矩阵表示.

很显然可以验证这样定义的线性映射具有一定的和谐性, 即 $\forall h(t) \in \mathbb{C}[t]$, 有 $m_{h(f)} = h(m_f)$. 只需逐一验证 $m_{f+g} = m_f + m_g$, $m_{fg} = m_f m_g$ 即可.

下面的定理给出了求解的方法:

定理13.23. *设 h_f 是 m_f 的极小多项式, $\forall \lambda \in \mathbb{C}$, I 是一零维理想, 则下面三个命题是等价的:*

1. $h_f(\lambda) = 0$,
2. λ 是 m_f 的特征值,
3. f 在 $V(I)$ 上取得值 λ, 即 $\lambda \in f(V(I))$.

证明. 由高等代数知识知道 1 和 2 等价是显然的.

$2 \Rightarrow 3$. 不妨设 $\lambda \notin f(V(I))$, 由特征值的定义知 $\exists \overline{z} \in A \setminus \{0\}$, 使得特征方程 $\overline{f - \lambda}\overline{z} = 0$ 成立. 令 $V(I) = \{p_1, \ldots, p_m\}$, 则 $f(p_i) \neq \lambda (\forall p_i \in V(I))$. 令 $g = f - \lambda$, 有 $g(p_i) \neq 0$, 设 $g_i (1 \leq i \leq m)$ 满足 $g_i(p_j) = \delta_{ij}$, 定义

$$g' = \sum_{i=1}^{m} \frac{1}{g(p_i)} g_i,$$

则 $g'(p_i)g(p_i) = 1 \Rightarrow 1 - g'g \in I(V(I)) = \sqrt{I} \Rightarrow \exists l \in \mathbb{N}((1 - g'g)^l \in I)$. 将其做二项式展开, 可得 g'' 满足 $1 - g''g \in I$, 即 $\overline{g''}\overline{g} = 1$, 故

$$\overline{f - \lambda}\overline{z} = 0 \Rightarrow \overline{g''}\overline{g}\overline{z} = 0 = \overline{z},$$

矛盾.

$3 \Rightarrow 1$. 设 $\lambda = f(p) (p \in V(I))$, 由 $h_f(m_f) = 0 \Rightarrow h_f(\overline{f}) = 0 \Rightarrow h_f(f) \in I$, 故 $h_f(\lambda) = h_f(f(p)) = 0$. □

推论13.8. *设 I 是零维理想, 则 m_{x_i} 的特征值集合 V_i 即为 $V(I)$ 中点的 x_i 分量, 且 $h_{x_i}(x_i)$ 是消元理想 $I \bigcap \mathbb{C}[x_i]$ 的唯一首一生成元.*

注181. 使用上面方法的好处在于, 当我们只用 Gröbner 基方法时, 由于要求指定的字典序下的 Gröbner 基, 其计算是较为复杂的. 而我们的定理仅涉及 A 的代数结构, 我们可以用计算量较小的分级字典序求出 Gröbner 基, 得到 A 上的基, 以此来进行特征值的计算 [9].

注182. 特征值法另一优点是符号求解时不不多次计算多项式组的 Gröbner 基即可得到各个消元理想 $I \bigcap \mathbb{C}[x_i]$ 的首一生成元, 从而精确求解出 x_i.

14 符号求和

我们接下来要讨论的符号求和问题是指符号表达式级数的求和, 它包括部分和问题以及无穷和(即部分和序列的极限)问题两部分. 如果求和上界 n 不出现在待求和的级数项中, 则称为不定求和, 这里我们只讨论不定求和的部分和问题.

14.1 多项式级数求和

为了解决多项式级数求和问题, 我们先来看一看单项式级数求和, 以下是我们熟知的几个求和公式, 公式的证明请参阅 [2].

定理14.1. *1.* $\sum\limits_{0\leq k<n} k = \frac{n(n-1)}{2}$

2. $\sum\limits_{0\leq k<n} k^2 = \frac{n(n-1)(2n-1)}{6}$

3. $\sum\limits_{0\leq k<n} k^3 = \frac{n^2(n-1)^2}{4}$

4. $\sum\limits_{0\leq k<n} k^4 = \frac{n(n-1)(2n-1)(3n^2-3n-1)}{30}$

注183. 这些公式容易用归纳法证明, 但很难直接从中找出显明的规律.

为了进一步地讨论, 我们引入差分算子和差分原函数的定义.

定义14.1 (差分算子)**.** 1. 平移算子 E 定义为

$$E(f)(x) = f(x+1),$$

另外定义 $E^n = E \circ E^{n-1}$, $E^1 = E$, $E^0 = e$.

2. 定义差分算子 $\Delta = E^1 - E^0$, 如果 $g = \Delta(f)$, 那么称 f 为 g 的差分原函数.

定理14.2.

$$\Delta(f \cdot g) = \Delta(f) \cdot g + f \cdot \Delta(g) - \Delta(f) \cdot \Delta(g).$$

证明.

$$\begin{aligned}\Delta(f \cdot g) &= E(f \cdot g) - f \cdot g \\ &= (E(f) - f) \cdot g + f \cdot (E(g) - g) - (E(f) - f) \cdot (E(g) - g) \\ &= \Delta(f) \cdot g + f \cdot \Delta(g) - \Delta(f) \cdot \Delta(g).\end{aligned}$$

□

定理14.3. 如果 g 是 f 的差分原函数, 即 $g = \Delta(f)$, 那么

$$\sum_{i=a}^{b-1} g(i) = f(b) - f(a).$$

证明.

$$\sum_{i=a}^{b-1} g(i) = \sum_{i=a}^{b-1} (f(i+1) - f(i)) = f(b) - f(a).$$

□

注184.

$$g = \Delta(f) \Leftrightarrow f = \Sigma(g),$$

这和微积分的互逆定理

$$g = D(f) \Leftrightarrow f = \int g$$

是类似的.

定义14.2 (函数降阶乘)**.** 函数 f 的 m 阶降阶乘

$$f^{\underline{m}} = f(x) \cdot f(x-1) \cdots f(x-m+1).$$

定理14.4.

$$\Delta(x^{\underline{m}}) = m x^{\underline{m-1}}.$$

证明.

$$\begin{aligned}\Delta(x^{\underline{m}}) &= (x+1)^{\underline{m}} - x^{\underline{m}} \\ &= ((x+1)-(x-m+1))x^{\underline{m-1}} \\ &= mx^{\underline{m-1}}.\end{aligned}$$

□

注185.

$$\sum_{i=0}^{n-1} i^{\underline{m}} = \frac{1}{m+1} n^{\underline{m+1}}.$$

设

$$x^m = \sum_{i=0}^{m} a_{m,i} x^{\underline{i}},$$

则

$$\begin{aligned}x^m &= x \cdot x^{m-1} \\ &= \sum_{i=0}^{m-1} a_{m-1,i}(x-i+i) \cdot x^{\underline{i}} \\ &= \sum_{i=0}^{m-1} a_{m-1,i} x^{\underline{i+1}} + \sum_{i=0}^{m-1} a_{m-1,i} i \cdot x^{\underline{i}} \\ &= x^{\underline{m}} + \sum_{i=1}^{m-1} (a_{m-1,i-1} + i \cdot a_{m-1,i}) x^{\underline{i}},\end{aligned}$$

于是有递推公式

$$a_{m,i} = a_{m-1,i-1} + i \cdot a_{m-1,i},$$

其中 $a_{m,0} = 1$.

$a_{m,i}$ 在组合数学中被称为第二类 Stirling 数(参见 [174]), 它表示将一个 m 元有限集划分为 i 个非空子集的方法数.

例14.1. 以 $m = 1, 2$ 为例.

1. $\sum_{i=0}^{n-1} i = \sum_{i=0}^{n-1} i^{\underline{1}} = \frac{1}{2} n^{\underline{2}} = \frac{1}{2} n(n-1)$,

2. $\sum_{i=0}^{n-1} i^2 = \sum_{i=0}^{n-1} (i^{\underline{2}} + i^{\underline{1}}) = \frac{1}{3} n^{\underline{3}} + \frac{1}{2} n^{\underline{2}} = \frac{1}{6} n(n+1)(2n+1)$.

定理14.5 (多项式级数部分和)**.** 设

$$g(x)=\sum_{m=0}^{d}g_m x^m,$$

则

$$\sum_{k=0}^{n-1}g(k)=\sum_{0\le i\le m\le d}g_m a_{m,i}\frac{n^{\underline{i+1}}}{i+1}.$$

证明. 根据注 185 有

$$\begin{aligned}\sum_{k=0}^{n-1}g(k)&=\sum_{k=0}^{n-1}(\sum_{m=0}^{d}g_m(\sum_{i=0}^{m}a_{m,i}k^{\underline{i}}))\\&=\sum_{0\le i\le m\le d}g_m a_{m,i}(\sum_{k=0}^{n-1}k^{\underline{i}})\\&=\sum_{0\le i\le m\le d}g_m a_{m,i}\frac{n^{\underline{i+1}}}{i+1}.\end{aligned}$$

□

14.2 超几何级数

和多项式级数的情形一样, 首先研究超几何单项式级数求和.

定义14.3 (超几何单项式)**.** 单项式级数通项 g_n 称为超几何单项式, 如果相邻项之比

$$r(n)=\frac{g_{n+1}}{g_n}$$

是关于 n 的有理函数.

例14.2. 二项式系数 $\binom{m}{n}$ 是超几何单项式, 因为

$$\begin{aligned}\frac{\binom{m}{n+1}}{\binom{m}{n}}&=\frac{\Gamma(m+1)\Gamma(n+1)\Gamma(m-n+1)}{\Gamma(n+2)\Gamma(m+n)\Gamma(m)}\\&=\frac{-n+m}{n+1}.\end{aligned}$$

设

$$r(n)=\frac{g_{n+1}}{g_n}=\frac{a(n)}{b(n)}\cdot\frac{c(n+1)}{c(n)},$$

并且满足 $\gcd(a(n), b(n+k)) = 1, \forall k \in \mathbb{N}$. 设 $\Delta(f) = g$, 则 $f_n = f_0 + \sum_{k=0}^{n-1} g_k$,

$$\frac{f_n}{g_n} = \frac{f_n}{f_{n+1} - f_n} = \frac{1}{\frac{f_{n+1}}{f_n} - 1}.$$

令 $y(n) = \frac{f_n}{g_n}$, 则

$$y(n+1)g(n+1) = f_{n+1} = (y(n)+1)g(n),$$

即 $r(n)y(n+1) = y(n) + 1$. 所以可取 $y(n) = \frac{b(n-1)}{c(n)}x(n)$, 其中 $x(n) \neq 0$ 且满足

$$a(n)x(n+1) - b(n-1)x(n) = c(n).$$

14.2.1 极大阶乘分解

极大阶乘分解(Greatest Factorial Factorization)是多项式的一种特殊分解.

定义14.4 (极大阶乘分解)**.** 设 f 为首一多项式, 记 f 的极大阶乘分解为

$$\mathrm{gff}(f) = \langle f_1, \cdots, f_m \rangle,$$

它满足:

1. $f = f_1^{\underline{1}} \cdots f_m^{\underline{m}}$.
2. $f_1, \cdots, f_m$ 为首一多项式且 $f_m \neq 1$.
3. $\gcd(f_i^{\underline{i}}, E(f_j)) = 1$.
4. $\gcd(f_i^{\underline{i}}, E^{-j} f_j) = 1, 1 \leq i, j \leq m$.

注186. 这样定义是为了保证

$$f_j^{\underline{j}} = f_j \cdot E^{-1} f_j \cdots E^{-j+1} f_j$$

没有更小的降阶乘因子, 因为如果 $g = \gcd(f_i^{\underline{i}}, E^{-j} f_j) \neq 1$, 则 $g^{\underline{j+1}} | f$.

关于极大阶乘分解有如下定理, 定理的证明请参阅 [174].

定理14.6. *1. 设 f 为首一多项式且 $f \neq 0$, 则 f 至多有一种极大阶乘分解.*

2. $\mathrm{gff}(\gcd(f, E(f))) = \langle f_2, \cdots, f_m \rangle$.

14.2.2 Gosper 算法

引理14.1. 给定超几何单项式 g, 设 $\sigma = \frac{E(g)}{g}$, $f = \tau \cdot g$, 则

$$\Delta(f) = g \Leftrightarrow E(\tau) \cdot \sigma - \tau = 1.$$

证明.

$$\begin{aligned}\Delta(f) &= E(f) - f \\ &= E(\tau) \cdot E(g) - \tau \cdot g \\ &= (E(\tau) \cdot \sigma - \tau)g.\end{aligned}$$

□

注187. 由超几何单项式的性质可以设 $\sigma = \frac{a}{b}$, $\tau = \frac{u}{v}$, 其中 $(a, b) = (u, v) = 1$, 并且 b, v 均为首一多项式, 上式化为

$$a \cdot v \cdot E(u) - b \cdot u \cdot E(v) = b \cdot v \cdot E(v).$$

这样我们就把求解差分原函数的问题归结为求解多项式方程. 理论上可以直接通过待定系数法解出这个方程, 但是计算量很大.

Gosper 算法利用极大阶乘分解来求解注 187 中的多项式方程.

记 $\gcd(E(f)) = \gcd(f, E(f))$, 设 $v_0 = \frac{v}{\gcd(E(v))}$, $v_1 = \frac{E(v)}{\gcd(E(v))}$, 则 $(v_0, v_1) = 1$, 方程两边同除以 $\gcd(E(v))$ 得

$$a \cdot v_0 \cdot E(u) - b \cdot v_1 \cdot u = b \cdot v_0 \cdot v_1 \cdot \gcd(E(v)),$$

其中

$$(u, v_0) = (E(u), v_1) = (u, v) = 1,$$

于是有

$$v_0 | b, v_1 | a.$$

设 $\mathrm{gff}(v) = \langle h_1, \cdots, h_m \rangle$, 则

$$\begin{aligned}v_0 &= \frac{h_1^1 h_2^2 \cdots h_m^m}{h_2^1 \cdots h_m^{m-1}} \\ &= h_1 \cdot E^{-1}(h_2) \cdots E^{-(m-1)}(h_m),\end{aligned}$$

$$\begin{aligned}v_1 &= \frac{E(h_1)^1 E(h_2)^2 \cdots E(h_m)^m}{h_2^1 \cdots h_m^{m-1}} \\ &= E(h_1) \cdot E(h_2) \cdots E(h_m),\end{aligned}$$

再由 $v_0|b$, $v_1|a$ 可得

$$h_i|E^{-1}(a), h_i|E^{i-1}(b).$$

若 $v \neq 1$, 则

$$\begin{aligned} 1 \neq h_m| \gcd(E^{-1}(a), E^{m-1}(b)) \\ = E^{-1}(\gcd(a(x), b(x+m))), \end{aligned}$$

而这等价于

$$\mathrm{res}_x(a(x), b(x+y))|_{y=m} = 0.$$

现在我们可以写出求 v 的某一个倍数 V 的 Gosper 算法.

算法14.1 (Gosper 算法).

输入:(a,b)=1, 其中 b 为首一多项式.

输出:首一多项式 V, 满足 $v|V$, 其中 $(u,v)=1$ 且

$$a \cdot v \cdot E(u) - b \cdot u \cdot E(v) = b \cdot v \cdot E(v).$$

1. 设 $R = \mathrm{res}_x(a(x), b(x+y))$, $d = \max\{k \in \mathbb{N}; k = 0 \text{ 或 } R(k) = 0\}$. 若 $d = 1$, 输出 1.
2. 设 $a_0 = a$, $b_0 = b$, 对 $i = 1, 2, \cdots, d$ 依次计算

$$H_i = \gcd(E^{-1}(a_{i-1}), E^{i-1}(b_{i-1})),$$

$$a_i = \frac{a_{i-1}}{E(H_i)}, b_i = \frac{b_{i-1}}{E^{-(i-1)}(H_i)}.$$

3. 输出 $H_1^1 \cdots H_d^d$.

注188. 不难看出 $h_i|H_i$, 因此

$$v = h_1^1 \cdots h_m^m | H_1^1 \cdots H_m^m \cdots H_d^d = V.$$

在算法实际运行过程中, 一旦 $a_i = 1$ 或 $b_i = 1$, 可以预计 $H_j = 1$, 其中 $j = i+1, \cdots, d$, 此时可以直接结束计算.

现在我们将方程化成了

$$a \cdot V \cdot E(U) - b \cdot E(V) \cdot U = b \cdot V \cdot E(V),$$

其中 V 为已知的首一多项式. 左右同除以 $g=\gcd(a\cdot V,b\cdot E(V))$, 得

$$r\cdot E(U)-s\cdot U=t,$$

其中 $r=\frac{a\cdot V}{g}$, $s=\frac{b\cdot E(V)}{g}$, $t=s\cdot V$.

设

$$U=U_e x^e+\cdots+U_0,$$

其中 e 可以通过如下定理确定, 定理的证明请参阅 [174].

定理14.7. *设 $m=\max\{\deg(r)-1,\deg(s-r)\}$, δ 为 $(s-r)$ 中 x^m 的系数.*

1. *若 $\deg(r)-1<\deg(s-r)$ 或 $\delta\notin\mathbb{N}$, 则 $e=\deg(t)-m$.*
2. *若 $\deg(t)-m=\delta$, 则多项式方程不可解.*
3. *若 $\deg(t)-m\neq\delta$, 则 $e=\max\{\deg(t)-m,\delta\}$, 如果 $e<0$, 则多项式方程不可解.*

注189. 对多项式方程比较对应项系数将得到一个上三角阵的线性方程组, 这时就可以解出 U 来. 通过这一系列手续, 最终求得

$$u=\frac{U}{\gcd(U,V)},v=\frac{V}{\gcd(U,V)}.$$

除了多项式级数与超几何级数的求和算法之外, 还有许多其他的求和算法. 例如 Wilf-Zeilberger 算法(参见 [138]), 它可以解决求和上界 n 出现在待求和的级数项中的超几何级数求和问题, 求出和函数所满足的递推方程, 因此也常常用于组合恒等式的自动证明. 再如针对有理函数级数的求和问题, 现有的算法包括 Moenck 算法, Horowitz 算法, Paule 算法(参见 [137])和 Karr 算法(参见 [100])等, Karr 算法的扩展版本还可以解决含根式的级数求和问题(参见 [101])以及嵌套求和问题(参见 [153]), 关于有理函数求和 更详细的介绍请参阅 [138] 和 [139].

15 符号积分

符号积分是符号计算的重要课题之一, 也是计算机代数系统最常用的核心功能之一. 众所周知, 对一个函数进行符号微分运算是较为简单的事情. 早在 20 世纪 50 年代计算机诞生之初, Kahrimanian[94] 和 Nolan[132] 就各自独立地写出了符号微分程序. 然而符号微分的反过程 — 符号积分相对而言却要复杂的多. 除了多项式的符号积分是直接的以外, 其他函数类的符号积分算法都是相当困难的. 1963 年, Slagle[162] 写出了第一个符号积分程序 SAINT(Symbolic Automatic INTegrator), SAINT 采用人工模拟求积分的启发性算法, 可以成功求解大部分大学一年级微积分考试试题. 1967 年, Moses[128], [129] 写出了 SIN(Symbolic INtegrator)程序, SIN 除了人工智能算法以外, 还了使用了一般性的 Risch 算法求解过程.

Risch 算法为符号积分理论的主要突破, 由 Risch 在他的一系列工作中所建立(包括 [147], [148], [149] 及其他一些未正式发表的论文), 为近几十年来符号积分理论发展奠定了基础. Risch 算法给出一个理论上确定性的过程, 用来判定一个给定函数是否有初等积分, 并在有初等积分时给出积分结果. Liouville 定理(定理 15.1)在 Risch 算法的构建中起到核心的作用.

尽管 Risch 算法在理论上十分优美, 但算法从来没有被完全地实现过. 原因是多方面的, 例如 Risch 算法要求完全分解多项式, 而多项式的因子分解往往是十分困难的; 在算法过程中可能会引入表示积分结果所不需要的代数数而使积分结果显得相当复杂; 代数函数的积分算法需要计算系数在基域代数闭包中, 指数为分数的幂级数, 因此较为繁琐; 被积函数和积分结果也不能包括特殊函数等等. 许多后

续的工作改进了这些缺陷, 并应用到实践中, 例如 Trager[21], Bertrand[30, 31] 在代数函数积分方面的工作, Cherry[52, 53], Knowles[102] 在特殊函数积分方面的工作等等.

本章主要介绍有理函数和初等超越函数的积分理论, 需要读者有抽象代数基础. 本章主要参考书为 [64], 和 [74]. 关于本章没有涉及的代数函数积分理论及其实现, 感兴趣的读者可以参考 Trager 的原始论文 [21], 文献 [74] 及其他相关文献.

15.1 有理函数积分

15.1.1 部分分式分解

最简单的多项式函数的积分算法是直接的. 我们下面考虑有理函数 $\dfrac{q(x)}{r(x)}$ 的符号积分, 其中 $q(x), r(x) \in \mathbb{C}[x]$, 满足最大公因子 $(q(x), r(x)) = 1$, 不失一般性还可假设 $\deg q(x) < \deg r(x)$. 求有理函数积分的最朴素的办法是利用部分分式分解.

引理15.1. 设 $(f(x), g(x)) = 1$, $\deg h(x) < \deg f(x)g(x)$, 则有

$$\frac{h(x)}{f(x)g(x)} = \frac{f_1(x)}{f(x)} + \frac{g_1(x)}{g(x)},$$

其中 $\deg f_1(x) < \deg f(x)$, $\deg g_1(x) < \deg g(x)$.

证明. 由 $(f(x), g(x)) = 1$ 知存在 $u(x)$, $v(x)$ 满足 $f(x)u(x) + g(x)v(x) = 1$, 于是

$$\frac{h(x)}{f(x)g(x)} = \frac{h(x)(f(x)u(x) + g(x)v(x))}{f(x)g(x)} = \frac{h(x)v(x)}{f(x)} + \frac{h(x)u(x)}{g(x)}.$$

令 $f_1(x) = h(x)v(x) \bmod f(x)$, $g_1(x) = h(x)u(x) \bmod g(x)$ 即得. □

设有因子分解 $r(x) = \prod_{i=1}^{n} (x - a_i)^{n_i}$, 反复利用引理 15.1 则有

$$\frac{q(x)}{r(x)} = \sum_{i=1}^{n} \frac{b_i(x)}{(x - a_i)^{n_i}},$$

其中 $\deg b_i(x) < n_i$. 再将 $b_i(x)$ 在 $x = a_i$ 处展开可得

$$\frac{q(x)}{r(x)} = \sum_{i=1}^{n} \sum_{j=1}^{n_i} \frac{b_{ij}}{(x - a_i)^j},$$

其中 b_{ij} 为常数. 于是得到

$$\int \frac{q(x)}{r(x)}\,dx = \sum_{i=1}^{n} b_{i1} \ln(x - a_i) - \sum_{i=1}^{n} \sum_{j=2}^{n_i} \frac{b_{ij}}{(j-1)(x - a_i)^{j-1}}. \tag{15.1}$$

由此有理函数符号积分的问题在理论上就被解决了, 即所有有理函数的积分均可由有理函数和对数函数表示, 分别称为积分的有理部分和对数部分. 但还有几个值得考虑的问题: 一方面我们需要在 $\mathbb{C}$ 上完全分解 $r(x)$, 而精确地做到这一点并不总那么容易; 另一方面在计算过程中可能会引入一些系数域扩域中的代数数常数, 而这些常数对于表达最终积分结果并不是必需的.

例15.1. 设 $q(x)=2x$, $r(x)=x^2+1\in\mathbb{Q}[x]$, 则 $\int \frac{q(x)}{r(x)}\,dx=\ln(x^2+1)$, 而在 $\mathbb{C}$ 上分解 $r(x)$ 需要引入额外的代数数 i.

15.1.2 Hermite 方法

Hermite 方法使用多项式的无平方分解来代替完全分解, 而无平方分解可以在系数域中完成(参见算法 9.9). 设 $r(x)$ 的无平方分解为 $r(x)=\prod_{i=1}^{m} r_i(x)^i$, 其中 $r_i(x)$ 无平方因子且两两互素. 从而

$$\frac{q(x)}{r(x)}=\frac{q(x)}{\prod_{i=1}^{m} r_i(x)^i}=\sum_{i=1}^{m}\frac{q_i(x)}{r_i(x)^i},$$

其中 $\deg q_i(x)<\deg r_i(x)^i$. 因此我们只需要考虑积分 $\int \frac{q_i(x)}{r_i(x)^i}\,dx$ 即可.

由于 $r_i(x)$ 无平方因子, 故 $(r_i(x),r_i'(x))=1$, 可设多项式 $a(x),b(x)$ 满足 $a(x)r_i(x)+b(x)r_i'(x)=1$, 代入进行分部积分得到

$$\begin{aligned}\int \frac{q_i(x)}{r_i(x)^i}\,dx&=\int \frac{q_i(x)(a(x)r_i(x)+b(x)r_i'(x))}{r_i(x)^i}\,dx\\&=\int \frac{q_i(x)a(x)}{r_i(x)^{i-1}}\,dx+\int \frac{q_i(x)b(x)r_i'(x)}{r_i(x)^i}\,dx\\&=\int \frac{q_i(x)a(x)+(q_i(x)b(x))'/(i-1)}{r_i^{i-1}}\,dx-\frac{1}{i-1}\frac{q_i(x)b(x)}{r_i^{i-1}}.\end{aligned}$$

为了简便起见, 我们将省略变元 x 与记号 dx, 以下的积分与导数总是对 x 而言的, 例如上式即可简写为

$$\int \frac{q_i}{r_i^i}=\int \frac{q_i a+(q_i b)'/(i-1)}{r_i^{i-1}}-\frac{1}{i-1}\frac{q_i b}{r_i^{i-1}}. \tag{15.2}$$

此过程称为 Hermite 约化过程. 每进行一次分部积分, 被积函数分母中 r_i 的次数便降低一次, 如此续行即可完全得到积分的有理部分.

15.1.3 Horowitz-Ostrogradsky 方法

Hermite 方法避免了完全分解, 但仍要求无平方分解和关于 r_i 的部分分式分解. Horowitz-Ostrogradsky 方法则不需要其他工具, 通过待定系数法将问题归结为线性方程的求解问题.

设

$$\int \frac{q}{r} = \frac{q_1}{r_1} + \int \frac{q_2}{r_2},$$

其中 $\dfrac{q_1}{r_1}$ 为积分的有理部分, 由部分分式的结果可知 $r_1 = (r, r')$, $r_2 = r/r_1$. 求导得到

$$\begin{aligned}\frac{q}{r} &= \left(\frac{q_1}{r_1}\right)' + \frac{q_2}{r_2} \\ &= \frac{q_1' r_1 - q_1 r_1'}{r_1^2} + \frac{q_2 r_1}{r} \\ &= \frac{q_1' r_2 - q_1 r_1' r_2 / r_1 + q_2 r_1}{r}.\end{aligned}$$

从而问题归结为求 q_1, q_2 满足 $q = q_1' r_2 - q_1 r_1' r_2 / r_1 + q_2 r_1$ 且 $\deg q_1 < \deg r_1$, $\deg q_2 < \deg r_2$. 以 q_1, q_2 的系数作为未知变量, 这是一个至多 $\deg r$ 维的线性方程组问题.

注190. 设 $\deg r = n$, 则 Hermite 方法的时间复杂度为 $O(M(n)\log n)$(参见 [174]), 其中 $M(n)$ 为多项式乘法的复杂度. 而 Horowitz-Ostrogradsky 方法的复杂度为 $O(n^3)$(解线性方程组), 尽管渐进意义上后者要比前者慢将近两个量级, 但在实践中还是要看具体情况而定.

注191. 文献 [45]2.2 节还提供了此原始 Hermite 约化过程的两个变体 — “二次版本”可以将约化步数降为为 $O(m^2)$, “线性版本”可以将约化步数降至 $O(m)$, 其中 m 为 r 的无平方因子的个数. “线性版本”还能够有效避免部分分式分解和无平方分解.

15.1.4 Rothstein-Trager 方法

下面的问题是如何求出对数部分, 设 $r(x)$ 为无平方因子的首一多项式, 则积分具有形式

$$\int \frac{q}{r} = \sum_{i=1}^{m} c_i \ln v_i,$$

其中 c_i 为互不相同的常数, v_i 为无平方因子的首一多项式, 两两互素(可以通过合并项来满足这些条件). 求导得到

$$\frac{q}{r}=\sum_{i=1}^{m}\frac{c_iv_i'}{v_i}.$$

由于 v_i 无平方因子且两两互素, 可知

$$r=\prod_{i=1}^{m}v_i.$$

记 $u_i=\prod\limits_{j\neq i}v_j$, 则有

$$r'=\sum_{i=1}^{m}u_iv_i'$$

及

$$q=\sum_{i=1}^{m}c_iu_iv_i'.$$

于是对于 $1\leq k\leq n$, 我们有

$$\begin{aligned}(q-c_kr',r)&=(q-c_kr',\prod_{i=1}^{m}v_i)\\&=\prod_{i=1}^{m}(q-c_kr',v_i)\\&=\prod_{i=1}^{m}\left(\sum_{j=1}^{m}(c_j-c_k)u_jv_j',v_i\right)\\&=v_k.\end{aligned}\tag{15.3}$$

最后一个等式成立是因为

$$((c_j-c_k)u_jv_j',v_i)=\begin{cases}v_i & i\neq j,\\1 & i=j,i\neq k,\\v_i & i=j,i=k,\end{cases}$$

从而

$$\left(\sum_{j=1}^{m}(c_j-c_k)u_jv_j',v_i\right)=\begin{cases}1 & i\neq k,\\v_i & i=k.\end{cases}$$

由式 (15.3) 可以看出, 当我们找出系数 c_k 后即可通过最大公因子的计算求出 v_k. 那么如何求出 c_k 呢? 通过同样的论证我们可以知道

$$(q-yr',r)>1\Leftrightarrow\exists 1\leq k\leq m,y=c_k,$$

而由推论 8.3 ,

$$(q-yr',r)>1 \Leftrightarrow \mathrm{Res}_x(q-yr',r)=0,$$

(此结式称为 Rothstein-Trager 结式)故求解关于 y 的方程 $\mathrm{Res}_x(q-yr',r)=0$ 即可得到所有的系数 c_k.

例15.2. 设 $r=x^3+x+1$, $q=3x^2+1$, 则 Rothstein-Trager 结式 $\mathrm{Res}_x(q-yr',r)=\mathrm{Res}_x((1-y)(3x^2+1),x^3+x+1)=-31(y-1)^3$. 从而方程 $\mathrm{Res}_x(q-yr',r)=0$ 有唯一的根 $y=1$, 最大公因子 $(q-yr',r)=r=x^3+x+1$, 故

$$\int \frac{q}{r}=\ln(x^3+x+1)$$

即为积分结果.

注192. Rothstein-Trager 方法的时间复杂度为 $O(nM(n)\log n)$[174].

注193. 对于高次 Rothstein-Trager 结式, 其根可能无法显式地表达出来. 设 r 无平方因子, n 个根为 $a_i(i=1,\dots,n)$, 不难求得 (15.1) 中的 b_{i1}, 此时可直接将积分表示为

$$\int \frac{q}{r}=\sum_{i=1}^{n}\frac{q(a_i)}{r'(a_i)}\ln(x-a_i),$$

便于进一步的运算.

注194. 对于实值函数的积分还应将 c_k 中互为相反数对应的对数项合并为更常见的 arctan 项.

15.1.5 Lazard-Rioboo-Trager 方法

Rothstein-Trager 方法解决了引入不必要的扩张常数的问题, 但仍需要计算许多代数扩域上多项式的最大公因子. Lazard-Rioboo-Trager 方法是对此的一个改进, 其过程只需要无平方分解和若干带余除法.

设 $r=\prod_{i=1}^{n}(x-a_i)$, $h=\mathrm{Res}_x(q-yr',r)$, 由结式的性质(参见 [12] 定理 3.14)得到

$$h=c\cdot\prod_{i=1}^{n}(q(a_i)-yr'(a_i)),$$

其中 c 为非零常数. 从而 h 中 c_k 的重数为满足 $q(a_i)-c_kr'(a_i)=0$ 的 i 的个数. 而

$$\begin{aligned}v_k&=(q-c_kr',r)\\&=(q-c_kr',\prod_{i=1}^{n}(x-a_i))\\&=\prod_{q(a_i)-c_kr'(a_i)=0}(x-a_i)\end{aligned}$$

可知 $\deg v_k$ 等于 h 中 c_k 的重数. 由此出发可知: 设被积函数的系数域为 F, 则只需要在 $F(y)[x]$ 中对 $q-yr'$ 和 r 做 Euclid 除法, 设 $W_i(x,y)\in F(y)[x]$ 为余式序列中关于 x 次数为 i 的多项式, 则有 $v_k=W_{\deg v_k}(x,c_k)$. 另外, 如果用多项式余式序列算法(PRS)(参见 8.3.2 节)来计算结式 h 的话, $W_i(x,y)$ 的计算可以与结式的计算同时进行.

15.2 Liouville 定理

从直观上来看, 所有有理函数总是可以积出. 这里"积出"的精确含义是, 可以找到一个初等函数作为被积函数的原函数. 为了将问题的本质看得更清楚, 我们引入如下定义.

定义15.1. 设 F 为一个特征为零的域, 映射 $D:F\to F$ 满足 $\forall f,g\in F$ 有

$$\begin{aligned} D(f+g)&=D(f)+D(g),\\ D(f\cdot g)&=f\cdot D(g)+g\cdot D(f). \end{aligned}$$

则 D 称为 F 上的微分算子, (F,D) 称为一个微分代数或者微分域. 集合 $C(F)=\{c\in F\mid D(c)=0\}$ 中的元素称为常数.

容易验证 D 拥有所有分析学教程中求导运算的基本性质, 我们在这里仅仅罗列而略去证明(参见 [45]).

命题15.1 (微分代数的基本性质). *1.* $\forall f\in F,c\in C(F)$, $D(cf)=cD(f)$.

2. (除法)$\forall f,g\in F,g\neq 0$,

$$D\left(\frac{f}{g}\right)=\frac{gD(f)-fD(g)}{g^2}.$$

3. $C(F)$ *为* F *的子域.*

4. $\forall f\in F^*,n\in\mathbb{Z}$, $D(f^n)=nf^{n-1}D(f)$.

5. (对数导数恒等式)$\forall f_1,\dots,f_n\in F^*,e_1,\dots,e_n\in\mathbb{Z}$,

$$\frac{D(f_1^{e_1}\cdots f_n^{e_n})}{f_1^{e_1}\cdots f_n^{e_n}}=e_1\frac{D(f_1)}{f_1}+\cdots+e_1\frac{D(f_n)}{f_n}.$$

6. (复合求导)设多项式 $P \in C(F)[x_1, \ldots, x_n]$, $f_1, \ldots, f_n \in F$, 则有

$$D(P(f_1, \ldots, f_n)) = \sum_{i=1}^{n} \frac{\partial P}{\partial x_i}(f_1, \ldots, f_n) D(f_i).$$

微分代数的基本性质使得我们可以完全脱离分析语言, 将符号积分的问题视为纯代数问题. 而我们关心的"初等函数", 也可以直接用代数语言来描述.

定义15.2 (初等函数)**.** 设 K 为一个微分代数, $K(\theta)$ 为 K 的微分代数扩张, 满足 $C(K(\theta)) = C(K)$, 微分算子 D 简记为'. θ 称为在 K 上的初等生成元, 若 θ 满足以下条件之一:

1. θ 在 K 上是代数的, 即有 $f(x) \in K[x]$ 使得 $f(\theta) = 0$;
2. θ 在 K 上是指数的, 即有 $\eta \in K$ 使得 $\dfrac{\theta'}{\theta} = \eta'$(此时记 $\theta = \exp \eta$);
3. θ 在 K 上是对数的, 即有 $\eta \in K$ 使得 $\theta' = \dfrac{\eta'}{\eta}$(此时记 $\theta = \ln \eta$).

称 $K(\theta_1, \ldots, \theta_n)$ 为 K 上的一个初等函数域, 若 $\forall 1 \leq i \leq n$, θ_i 为 $K(\theta_1, \ldots, \theta_{i-1})$ 上的初等生成元(约定 $K(\theta_0) = K$). 任意一个 K 上的初等函数域中的元素称为 K 上的初等函数.

注195. 考虑 $K = \mathbb{Q}(x)$, $\mathbb{R}(x)$ 或 $\mathbb{C}(x)$ 时, 以上定义的就是分析教程中通常意义下"初等函数".

注196. 定义中的 $C(K(\theta)) = C(K)$ 保证了如果 $\theta' \in K$ 的话, θ' 必定不是 K 中元素的导数.

下面来自 Liouville 的重要定理, 描述了积分为初等函数时必定具有的形式. 定理证明需要对三种特殊扩张情形进行讨论, 并最终得出一般情(参见形 [74], [45]).

定理15.1 (Liouville)**.** *设 K 为一个微分代数, $f \in K$, 若 $\int f$ 为 K 上的初等函数(即 f 有初等积分), 则有*

$$\int f = v_0 + \sum_{i=1}^{n} c_i \ln v_i,$$

其中 $v_0 \in K$, $c_i \in \overline{C(K)}$(代数闭包), $v_i \in \hat{K} = K(c_1, \ldots, c_n)$.

注197. 可以验证在 $K = \mathbb{Q}(x)$, $\mathbb{R}(x)$ 或 $\mathbb{C}(x)$ 时与我们在上一节的有理函数积分结果是一致的.

15.3 超越对数函数积分

15.3.1 分解引理

设 θ 为 K 上的一个超越对数函数, 我们考虑 $f \in K(\theta)$ 的积分问题. 可设 $f = p + \dfrac{q}{r}$, 其中 $p, q, r \in K[\theta]$, q, r 互素且 $\deg q < \deg r$, 下面的引理使得我们能够沿着和有理函数积分类似的道路走下去.

引理15.2 (超越对数函数分解引理). *设 f 为 K 上的超越对数函数, 若 f 在 K 上有初等积分, 则 p, $\dfrac{q}{r}$ 都在 K 上有初等积分.*

证明. 以 $K(\theta)$ 代替 K 应用 Liouville 定理可得

$$\int f = v_0 + \sum_{i=1}^{n} c_i \ln v_i,$$

其中 $v_0 \in K(\theta)$, $c_i \in \overline{C(K(\theta))} = \overline{C(K)}$, 记 $\hat{K} = K(c_1, \ldots, c_n)$, 通过拆分对数项可以保证 $v_i \in \hat{K}[\theta]$ 为首一多项式. 求导得

$$f = v_0' + \sum_{i=1}^{n} c_i \frac{v_i'}{v_i}.$$

设 $v_0 = p_0 + \dfrac{q_0}{r_0}$, 其中 $p_0, q_0, r_0 \in K[\theta]$, q_0, r_0 互素且 $\deg q_0 < \deg r_0$. 根据 v_i 为 $\hat{K}$ 中元素还是 $\hat{K}[\theta]$ 中正次数多项式将右端第二项拆为两部分的求和项, 可知

$$\begin{aligned} p + \frac{q}{r} &= \left(p_0 + \frac{q_0}{r_0}\right)' + \sum_{i=1}^{k} c_i \frac{v_i'}{v_i} + \sum_{i=k+1}^{n} c_i \frac{v_i'}{v_i} \\ &= \underbrace{\left(p_0' + \sum_{i=1}^{k} c_i \frac{v_i'}{v_i}\right)}_{\in \hat{K}[\theta]} + \underbrace{\left(\left(\frac{q_0}{r_0}\right)' + \sum_{i=k+1}^{n} c_i \frac{v_i'}{v_i}\right)}_{\in \hat{K}(\theta)\text{为真分式}}. \end{aligned}$$

由于 $\theta' = \dfrac{\eta'}{\eta}$, 故 $\hat{K}[\theta]$ 中的多项式的导数仍为 $\hat{K}[\theta]$ 中的多项式. 由 $v_i \in \hat{K}$, 可知右端的第一部分为 $\hat{K}[\theta]$ 中的多项式; 又由 v_i 为 $\hat{K}[\theta]$ 中首一多项式, 可验证 $\dfrac{v_i'}{v_i}$ 为 $\hat{K}(\theta)$ 中的真分式.

由于 θ 在 K 上超越, 等式两端的关于 θ 的多项式和真分式必定对应相等, 从而

$$\begin{cases} p = p_0' + \displaystyle\sum_{i=1}^{k} c_i \frac{v_i'}{v_i}, \\ \dfrac{q}{r} = \left(\dfrac{q_0}{r_0}\right)' + \displaystyle\sum_{i=k+1}^{n} c_i \frac{v_i'}{v_i}. \end{cases} \tag{15.4}$$

故 p, $\dfrac{q}{r}$ 都在 K 上有初等积分. □

15.3.2 多项式部分

由分解引理, 我们可以等价地分别对多项式部分 p 和有理部分 $\dfrac{q}{r}$ 分别求积分, 如果其中有一个无初等积分, 则 f 也必无初等积分.

注198. 一般来说将被积函数 f 拆成两项分别积分与直接积分并不等价. 例如

$$\int x^x, \quad \int x^x \ln x$$

均非初等函数, 但

$$\int x^x(1+\ln x) = x^x.$$

注199. 在下面的算法过程中, 为了求得 $K(\theta)$ 中函数的积分, 我们预先需要知道如何对基域 K 中的函数作积分. 例如我们已经知道有理函数域 $K=\mathbb{C}(x)$ 上的积分方法, 因此便可以解决例如 $\mathbb{C}(x,\ln x)$ 中的积分问题. 再取 $\mathbb{C}(x,\ln x)$ 作为新的基域 K, 便能得到 $\mathbb{C}(x,\ln x,\ln\ln x)$ 中函数的积分方法. 如此可以递归的地做下去, 构成了一种“塔状”的求积分过程, 我们在后面可以看到(参见定理 16.1), 类似这样的过程最终可以在理论上解决任意复杂形式初等函数积分问题.

在式 (15.4) 中设

$$p=\sum_{i=0}^{m} A_i\theta^i, \quad p_0=\sum_{i=0}^{n} B_i\theta^i,$$

其中 $A_i, B_i \in K$, 则有

$$\sum_{i=1}^{m} A_i\theta^i = \sum_{i=0}^{n} B_i'\theta^i + \sum_{i=1}^{n} iB_i\theta^{i-1}\theta' + \sum_{i=1}^{k} \frac{c_i v_i'}{v_i}.$$

由于 A_i 已知, 我们的目标是求出 B_i 和 $\sum\limits_{i=1}^{k} c_i \ln v_i$. 利用 θ 在 K 上超越, 可得

$$n=\begin{cases} m+1 & \text{若 } B_n'=0, \\ m & \text{若 } B_n'\neq 0. \end{cases}$$

若 $B_n'=0$, 则 B_{m+1} 为非零常数, 否则可约定 $B_{m+1}=0$, 于是无论何种情况考虑 θ 的最高次项 θ^m 可得

$$A_m = B_m' + (m+1)B_{m+1}\theta',$$

所以

$$B_m = -(m+1)B_{m+1}\theta + \int A_m.$$

因此若 p 在 K 上初等可积, 则必有 $\int A_m$ 为 $\hat{K}[\theta]$ 中次数不超过一次的多项式, 进一步可确定下常数 B_{m+1} 以及 $B_m \in K$. 当然, 由于 $\int A_m$ 积分常数的存在, B_m 的确定是在相差一个常数下而言的, 设此积分常数为 b_m. 接下来再考虑两边 θ^{m-1} 项可得

$$A_{m-1} = B'_{m-1} + m(B_m + b_m)\theta',$$

同样可知

$$B_{m-1} = -mb_m\theta + \int A_{m-1} - mB_m\theta',$$

于是若 p 在 K 上初等可积, 则 $\int A_{m-1} - mB_m\theta'$ 为 $\hat{K}[\theta]$ 中次数不超过一次的多项式, 可确定常数 b_m 及 $B_{m-1} + b_{m-1}(b_{m-1}$ 为积分常数). 如此续行, 直到比较 θ^0 项可得

$$A_0 = B'_0 + (B_1 + b_1)\theta' + \sum_{i=1}^{k} \frac{c_i v'_i}{v_i},$$

积分得

$$B_0 + \sum_{i=1}^{k} c_i \ln v_i = \int A_0 - \int (B_1 + b_1)\theta'.$$

最终求出常数 b_1, 以及 K 上的对数部分 $\sum\limits_{i=1}^{k} c_i \ln v_i$(注意这最后一步右端积分未必落在 K 中), 从而我们(在相差一个常数的意义下)求出了多项式部分的积分.

15.3.3 有理部分与对数部分

我们希望求出 (15.4) 中的 $\dfrac{q_0}{r_0}$ 和 $\sum\limits_{i=k+1}^{n} c_i \ln v_i$, 求积分的有理部分和对数部分可以和有理函数积分类似地做. 为确保类似 Hermite 约化的过程能够进行, 我们验证如下的引理.

引理15.3. 设 $r \in K[\theta]$ 为正次数的无平方因子的多项式, 则 $(r, r') = 1$.

证明. 设 $r = \prod\limits_{i=1}^{n} (\theta - a_i)$ 为 r 在 $\overline{K}$ 上的分解. 则由复合函数求导

$$\begin{aligned} r' &= \sum_{i=1}^{n} \prod_{j \neq i} (\theta - a_i)(\theta - a_i)' \\ &= \sum_{i=1}^{n} \prod_{j \neq i} (\theta - a_i)(\frac{\eta'}{\eta} - a'_i). \end{aligned}$$

若 $(r,r')\neq 1$, 则存在 i, 使得 $\theta-a_i\mid \dfrac{\eta'}{\eta}-a_i'$. 由于 $\dfrac{\eta'}{\eta}-a_i'\in\overline{K}$, 而 θ 在 K 上超越, 必有 $\dfrac{\eta'}{\eta}=a_i'$. 于是 $\theta=a_i+c(c$ 为常数)不在 K 上超越, 矛盾! □

有了引理 15.3 中 $(r,r')=1$ 的保证, 我们就可以施行 Hermite 约化过程了. 将 r 在 $K[\theta]$ 上作无平方分解 $r=\prod\limits_{i=1}^{n}r_i^i$, 完全类似可以得到式 (15.2) 的约化结果, 从而求出积分的有理部分 $\dfrac{q_0}{r_0}$.

同样有求对数部分的 Rothstein-Trager 方法的类似版本. 设 $r\in K[\theta]$ 无平方因子, 则积分的形式必为 $\displaystyle\int\frac{q}{r}=\sum_{i=k+1}^{n}c_i\ln v_i$, 其中 $c_i\in\overline{C(K)}$ 为常数, $v_i\in\hat{K}[\theta]$. 求得 Rothstein-Trager 结式 $\mathrm{Res}_\theta(q-yr',r)$ 关于 y 的根 $c_i\in\overline{K(\theta)}$, 若有 c_i 不是常数, 则 $\dfrac{q}{r}$ 在 $K(\theta)$ 上必无初等积分; 若 c_i 均为常数, 计算最大公因子 $(q-c_ir',r)$ 即可得到 v_i.

例15.3. 计算 $\displaystyle\int\frac{1}{x\ln x}$.

令 $\theta=\ln x$, 为 $K=\mathbb{Q}(x)$ 上的超越对数函数, 被积函数为 $\dfrac{q}{r}=\dfrac{1}{x\cdot\theta}\in K(\theta)$. r 无平方因子, 可以直接施行 Rothstein-Trager 方法. $r'=\theta+1$, 故

$$\begin{aligned}\mathrm{Res}_\theta(q-yr',r)&=\mathrm{Res}_\theta(1-y(\theta+1),x\theta)\\&=\begin{vmatrix}-y & x\\ 1-y & 0\end{vmatrix}\\&=-(1-y)x.\end{aligned}$$

可知此结式关于 y 只有一个根 $c=1$, 且为常数. 计算最大公因子

$$\begin{aligned}v=(q-cr',r)&=(1-(\theta+1),x\theta)\\&=(\theta,x\theta)\\&=\theta.\end{aligned}$$

最终可知

$$\int\frac{1}{x\ln x}=c\ln v=\ln\theta=\ln\ln x$$

即为积分结果.

例15.4. 计算 $\displaystyle\int\frac{1}{\ln x}$.

令 $\theta=\ln x$, 为 $K=\mathbb{Q}(x)$ 上的超越对数函数, 被积函数为 $\dfrac{q}{r}=\dfrac{1}{\theta}\in K(\theta)$. r 无平方因子, 可以直接施行 Rothstein-Trager 方法. $r'=\dfrac{1}{x}$, 故 $\mathrm{Res}_\theta(q-yr',r)=$

$\mathrm{Res}_\theta(1-y\cdot\frac{1}{x},\theta)=1-\frac{y}{x}$, 结式关于 y 的根为 $c=x$ 非常数, 可知 $\frac{1}{\ln x}$ 不是初等可积的.

15.4 超越指数函数积分

我们沿着有理函数积分的道路顺利地解决了超越对数函数积分的问题, 接下来自然要考虑的是超越指数函数积分了, 即考虑当 θ 为 K 上的超越指数函数时 $f\in K(\theta)$ 的积分问题.

但这个问题变得要复杂一些, 一方面在计算过程中我们需要在基域 K 中求解微分方程(被称为 Risch 微分方程[147])

$$y'+f\cdot y=g,$$

其中 $f,g\in K$, $y\in K$ 为未知函数. 我们将在微分方程符号解 一章中专门讨论 Risch 微分方程, 而且将看到 Risch 微分方程和符号积分在某种意义下的相互"纠缠". 现在我们暂时假设已经能够在 K 上完成这样的求解.

复杂性的另一面在于, 即使 $r\in K[\theta]$ 无平方因子也不能保证 $(r,r')=1$.

例15.5. 考虑 $\theta=e^x$, $r=\theta$, r 无平方因子, 但 $(r,r')=(\theta,\theta)=\theta\neq 1$.

因此我们需要对引理 15.3 做一个小的修订.

引理15.4. 设 $r\in K[\theta]$ 为正次数的无平方因子的多项式, 且 $\theta\nmid r$, 则 $(r,r')=1$.

证明. 设 $r=\prod_{i=1}^{n}(\theta-a_i)$ 为 r 在 $\overline{K}$ 上的分解. 则由复合函数求导

$$\begin{aligned}r'&=\sum_{i=1}^{n}\prod_{j\neq i}(\theta-a_i)(\theta-a_i)'\\&=\sum_{i=1}^{n}\prod_{j\neq i}(\theta-a_i)(\eta'\theta-a_i').\end{aligned}$$

若 $(r,r')\neq 1$, 则存在 i, 使得 $\theta-a_i\mid\eta'\theta-a_i'$, 从而 $\eta'a_i=a_i'$. 由于 $\theta\nmid r$, 故 $a_i\neq 0$. 而

$$\left(\frac{a_i}{\theta}\right)'=\frac{a_i'\theta-\theta'a_i}{\theta^2}=\frac{a_i'-\eta'a_i}{\theta}=0,$$

可知 $a_i=c\theta$(c 为非零常数), 这与 θ 在 K 上超越矛盾! □

15.4.1 分解引理

根据引理 15.4 , 为了能够继续使用 Hermite 约化, 我们需要将分母中的 θ 因子分离出来, 这就引导我们引入如下定义.

定义15.3 (广义多项式). $K(\theta)$ 中形如 $\sum\limits_{i=-m}^{n} A_i\theta^i$ 的元素称为 $K(\theta)$ 中的广义多项式.

在这样的定义下, f 有广义分解 $p+\dfrac{q}{r}$, 其中 p 为广义多项式, $q,r\in K[\theta]$, q, r 互素, $\deg q<\deg r$ 且有 $\theta\nmid r$. 有类似于引理 15.2 的分解引理, 证明过程也是类似的.

引理15.5 (超越指数函数分解引理). *设 f 为 K 上的超越指数函数, 若 f 在 K 上有初等积分, 则 p 与 $\dfrac{q}{r}$ 都在 K 上有初等积分.*

证明. 以 $K(\theta)$ 代替 K 应用 Liouville 定理可得

$$\int f=v_0+\sum_{i=1}^{n} c_i\ln v_i,$$

其中 $v_0\in K(\theta)$, $c_i\in\overline{C(K(\theta))}=\overline{C(K)}$, 记 $\hat{K}=K(c_1,\dots,c_n)$, 通过拆分对数项可以保证 $v_i\in\hat{K}[\theta]$ 为首一多项式. 求导得

$$f=v_0'+\sum_{i=1}^{n} c_i\frac{v_i'}{v_i}.$$

设 $v_0=p_0+\dfrac{q_0}{r_0}$ 为广义分解, 根据 v_i 为 $\hat{K}$ 中元素还是 $\hat{K}[\theta]$ 中正次数多项式将右端第二项拆为两部分. 我们要注意 $\hat{K}[\theta]$ 中多项式求导的特殊性, 由于

$$(A_i\theta^i)'=A_i'\theta^i+A_ii\theta^{i-1}\theta'=(A_i'+A_ii\eta')\theta^i,$$

因此多项式求导后每个单项的次数都不变, 故可不妨假定 $\theta\nmid v_i$. 由于 $\dfrac{v_i'}{v_i}$ 不再是真分式, 因此还需将 v_i' 中的最高次项分离出来(例如减去 $n_i\eta' v_i$, 其中 $\deg v_i=n_i$), 则可得

$$\begin{aligned}p+\frac{q}{r}&=\left(p_0+\frac{q_0}{r_0}\right)'+\sum_{i=1}^{k} c_i\frac{v_i'}{v_i}+\sum_{i=k+1}^{n} c_i\frac{v_i'}{v_i}\\&=\underbrace{\left(p_0'+\sum_{i=1}^{k} c_i\frac{v_i'}{v_i}+\sum_{i=k+1}^{n} c_in_i\eta'\right)}_{\in\hat{K}(\theta)\text{为广义多项式}}+\underbrace{\left(\left(\frac{q_0}{r_0}\right)'+\sum_{i=k+1}^{n} c_i\frac{v_i'-n_i\eta' v_i}{v_i}\right)}_{\in\hat{K}(\theta)\text{为真分式, }\theta\nmid v_i}\end{aligned}$$

由于 θ 在 K 上超越, 等式两端的关于 θ 的广义多项式和真分式必定对应相等, 从而

$$\begin{cases} p = p_0' + \sum_{i=1}^{k} c_i \dfrac{v_i'}{v_i} + \sum_{i=k+1}^{n} c_i n_i \eta', \\ \dfrac{q}{r} = \left(\dfrac{q_0}{r_0}\right)' + \sum_{i=k+1}^{n} c_i \dfrac{v_i' - n_i \eta' v_i}{v_i}. \end{cases}$$

故

$$\begin{cases} \int p = p_0 + \sum_{i=1}^{k} c_i \ln v_i + \sum_{i=k+1}^{n} c_i n_i \eta, \\ \int \dfrac{q}{r} = \dfrac{q_0}{r_0} + \sum_{i=k+1}^{n} c_i (\ln v_i - n_i \eta), \end{cases} \tag{15.5}$$

都在 K 上有初等积分. □

15.4.2 多项式部分

在式 (15.5) 中设

$$p = \sum_{i=-m}^{n} A_i \theta^i, \quad p_0 = \sum_{i=-m'}^{n'} B_i \theta^i,$$

其中 $A_i, B_i \in K$, 则有

$$\sum_{i=-m}^{n} A_i \theta^i = \left(\sum_{i=-m'}^{n'} B_i \theta^i \right)' + \sum_{i=1}^{k} \frac{c_i v_i'}{v_i} + \sum_{k+1}^{n} c_i n_i \eta'.$$

由于 A_i 已知, 我们的目标是求出 B_i 和 $\sum_{i=1}^{k} c_i \ln v_i + \sum_{i=k+1}^{n} c_i n_i \eta$. 由于多项式求导不改变次数, 故必有 $n = n'$, $m = m'$. 比较对应项系数得到

$$A_i = B_i' + i B_i \eta', \quad i = -m, \ldots, n, \ i \neq 0. \tag{15.6}$$

这些是 K 上的关于未知量 B_i 的 Risch 方程, 我们假定了能够在 K 上求解此类方程或给出无解的结论. 若其中有一个方程无解, 则 f 在 $K(\theta)$ 上无初等积分; 否则可求出 B_i $(i = -m, \ldots, n, \ i \neq 0)$, 最后根据 $\int A_0$ 求出 $B_0 + \sum_{i=1}^{k} c_i \ln v_i + \sum_{i=k+1}^{n} c_i n_i \eta$.

15.4.3 有理部分和对数部分

由于我们有了引理 15.4 的保证, 依然可以对广义分解中的 $\dfrac{q}{r}$ 进行 Hermite 约化过程, 从而得到积分的有理部分.

同样可以用类似于 Rothstein-Trager 的方法计算对数部分, 设 r 无平方因子, 则 $\dfrac{q}{r}$ 形如 $\sum\limits_{i=k+1}^{n} c_i \dfrac{v_i' - n_i\eta' v_i}{v_i}$, 可知 $r = \prod\limits_{j=k+1}^{n} v_j$. 记 $u_i = \prod\limits_{j\neq i} v_j$, 则

$$r' = \sum_{j=k+1}^{n} u_j v_j',$$

及

$$q = \sum_{j=k+1}^{n} c_j (v_j' - n_j\eta' v_j) u_j.$$

于是

$$\begin{aligned}
&\left(q - c_i\left(r' - r\sum_{j=k+1}^{n} n_j\eta'\right), r\right)\\
&= \left(\sum_{j=k+1}^{n} (c_j - c_i)(v_j' - n_j v_j\eta')u_j, r\right)\\
&= \prod_{l=k+1}^{n}\left(\sum_{j=k+1}^{n} (c_j - c_i)(v_j' - n_j v_j\eta')u_j, v_l\right)\\
&= v_i.
\end{aligned}$$

因为 $\sum\limits_{j=k+1}^{n} n_j = \deg r$, 所以可以通过求结式 $\mathrm{Res}_\theta(q - y(r' - r\deg r\cdot\eta', r)$ 关于 y 的根来求得 c_j. 与超越对数情形类似, 如果此结式有根 c_i 非常数, 则 $\dfrac{q}{r}$ 在 $K(\theta)$ 上无初等积分; 若所有根均为常数, 则通过计算最大公因子 $(q - c_i(r' - r\deg r\cdot\eta'), r)$ 即可求得 v_i.

16 微分方程符号解

微分方程在数学, 自然科学乃至社会科学中的都起着重要作用, 求解微分方程的方法被广泛地发展起来. 在计算机代数中, 求出微分方程的符号解也是一个令人众多研究者着迷的课题. 微分 Galois 理论, Lie 对称理论等众多理论被发展并应用在此领域. 关于这方面更详细的综述和文献目录, 我们推荐读者参考 [158] 和 [80] 的 2.11 节.

本章主要介绍微分方程符号解中最基本的问题, 包括 Risch 微分方程, 一阶线性微分方程, 微分 Galois 理论简介, 二阶微分方程的 Kovacic 算法, 高阶线性微分方程的多项式解, 有理解与指数解以及二阶微分方程的特殊解, 主要参考书为 [62], [105] 和 [170]. 阅读本章需要读者具备基本的微分方程, 抽象代数知识(例如 [11], [6]), 对于 Laurent 展式, 极点, Puiseux 展式等概念有所了解.

16.1 Risch 微分方程

在考虑超越指数函数积分时我们遇到了形如

$$A_i(x) = B_i(x)' + iB_i(x)\eta'(x)$$

的微分方程 15.6 , 其中数 $A_i, \eta' \in K$(基域)为已知函数, $B_i \in K$ 为未知函数. 这是一个 Risch 微分方程的求解问题. 一般地, 设 $f, g \in K$, Risch 微分方程问题就是要判断

$$y' + fy = g \tag{16.1}$$

在 K 中是否有解, 如果有解则构造出解. 此问题是最简单的一类微分方程, 同时对于符号积分的求解过程也是本质上重要的.

16.1.1 有理函数域

首先我们考虑最简单的有理函数域情形, 即设 $K=\mathbb{Q}(x)$, $\mathbb{R}(x)$ 或 $\mathbb{C}(x)$. 记 $\operatorname{den}(y)$ 表示有理函数 y 的既约分母, 对于任意不可约多项式 $p\in K$ 且 $p\mid\operatorname{den}(y)$, 我们希望求出 p 的次数. 设 $p^{\alpha}\|\operatorname{den}(y)$, $p^{\beta}\|\operatorname{den}(f)$, $p^{\gamma}\|\operatorname{den}(g)$. 可令 $y=\dfrac{v}{p^{\alpha}u}$, 其中 $(p,u)=1$, 则 $y'=\dfrac{v'pu-v(pu'+\alpha p'u)}{p^{\alpha+1}u^2}$, 由 $(p,p')=1$ 可知 $p^{\alpha+1}\|\operatorname{den}(y')$. 故方程 (16.1) 中三项 y', fy, g 的分母中 p 的次数分别为 $\alpha+1$, $\alpha+\beta$, γ. 比较两端分母中 p 的次数, 分为三种情形:

1. $\beta>1$ 时, 必有 $\gamma=\alpha+\beta$;
2. $\beta<1$ 时, 必有 $\gamma=\alpha+1$;
3. $\beta=1$ 时,
   - 若左端求和后无约分, 则 $\gamma=\alpha+1$;
   - 若左端求和后有约分. 由 $(p,u)=1$ 可找到 a, b, 使得 $ap^{\alpha}+bu=1$, 从而
   $$y=\frac{v}{p^{\alpha}u}=\frac{v(ap^{\alpha}+bu)}{p^{\alpha}u}=\frac{vb}{p^{\alpha}}+\frac{va}{u}=:\frac{Y}{p^{\alpha}}+\tilde{y},$$
   满足 $\deg Y<\deg p$, $p^{\alpha}\nmid\operatorname{den}(\tilde{y})$. 同理可设 $f=\dfrac{F}{p}+\tilde{f}$ 满足 $\deg F<\deg p$, $p\nmid\operatorname{den}(\tilde{f})$. 从而

$$y'+fy=-\frac{\alpha Yp'}{p^{\alpha+1}}+\frac{Y'}{p^{\alpha}}+\tilde{y}'+\frac{FY}{p^{\alpha+1}}+\frac{F\tilde{y}}{p}+\frac{Y\tilde{f}}{p^{\alpha}}+\tilde{f}\tilde{y},$$

由于有约分, 左端求和分母为 $p^{\alpha+1}$ 的项之和必定可以约分, 于是

$$p\mid-\alpha Yp'+FY,$$

而 p 不可约, $\deg Y<\deg p$, 故 $(p,Y)=1$, 可得

$$p\mid-\alpha p'+F.$$

又由 $\deg F<\deg p$ 可知必有 $F=\alpha p'$, 即 $\alpha=\dfrac{F}{p'}$.

由此得到了未知函数 y 分母中 p 的次数 α 的上界, 使我们便于使用待定系数法. 通过上面的讨论, 我们实际上证明了 [147] 中的如下引理.

引理16.1. 若 $\dfrac{F}{p'}$ 为整数且 $\beta=1$, 则

$$\alpha \le \max\left\{\gamma-1, \frac{F}{p'}\right\},$$

否则

$$\alpha \le \min\{\gamma-1, \gamma-\beta\}.$$

仔细验证可以发现, 只有在论证 $(p,Y)=1$ 时用到了 p 为不可约多项式, 其他地方均可以将条件减弱为 p 为无平方因子的多项式. 因此可以首先用 $\mathrm{den}(f),\mathrm{den}(g)$ 的无平方因子分解代替完全分解. 需要处理的是对幂次 $\beta=1$ 的无平方因子 p, 求出 p 的不可约因子 q_i, 使得 $z=\dfrac{F}{q_i'}$ 为整数. 与 Rothstein-Trager 方法类似得论证可以得到

$$F=zq_i' \Leftrightarrow (F-zp',p)\text{非平凡} \Leftrightarrow \mathrm{Res}_x(F-zp',p)=0.$$

从而我们只需求结式 $\mathrm{Res}_x(F-zp',p)=0$ 关于 z 的整数根即可. 我们得到了如下计算 $\mathrm{den}(y)$ 的一个倍数的方法.

算法16.1 (计算 $\mathrm{den}(y)$ 的一个倍数)**.**

1. 计算无平方分解 $\mathrm{den}(f)=\prod_{i=1}^{n} p_i^{\beta_i}$, $\mathrm{den}(g)=\prod_{i=1}^{n} p_i^{\gamma_i}$, 其中 $\beta_i,\gamma_i\ge 0$, p_i 无平方因子.
2. 对每个 $1\le i\le n$, 令 $p=p_i$, $\beta=\beta_i$, $\gamma=\gamma_i$.
   - 若 $\beta\ne 1$, 令 $\alpha=\min\{\gamma-1,\gamma-\beta\}$, $r=p^{\alpha}$;
   - 否则计算分解 $f=\dfrac{F}{p}+\tilde{f}$, 求出 $\mathrm{Res}_x(F-zp',p)=0$ 的整数根 $z_1,\dots,z_m$(例如利用算法 9.7). 令 $\alpha_j=\max\{\gamma-1,z_j\}$, $q_j=(F-z_jp',p)$, $r=p^{\gamma-1}\cdot\prod_{j=1}^{m} q_j^{\alpha_j-(\gamma-1)}$.
3. 累乘所有 r 并输出, 算法终止.

在得到 $\mathrm{den}(y)$ 的一个倍数之后, 只要确定 y 分子次数的一个上界, 即可使用待定系数法求出 y 的分子来了. 设 $y=\dfrac{a}{b}$, 我们希望求出 $\alpha=\deg a$. 将其带入方程 (16.1) 并两端乘以最小公分母可以得到形如

$$Ra'+Sa=T$$

的等式, 其中 R, S, T 为多项式, 记 $\beta = \deg R$, $\gamma = \deg S$, $\delta = \deg T$, 则 Ra', Sa, T 的次数分别为 $\beta + \alpha - 1$, $\gamma + \alpha$, δ. 比较等式两端的次数, 分为三种情形:

1. $\gamma < \beta - 1$ 时, 必有 $\delta = \beta + \alpha - 1$;
2. $\gamma > \beta - 1$ 时, 必有 $\delta = \gamma + \alpha$;
3. $\gamma = \beta - 1$ 时,
   - 若左端求和后最高项未被消去, 则 $\delta = \gamma + \alpha$;
   - 若左端求和后最高项被消去. 设 $a = \sum_{i=1}^{\alpha} a_i x^i$, $R = \sum_{i=1}^{\beta} r_i x^i$, $S = \sum_{i=1}^{\gamma} s_i x^i$, 则有 $\alpha r_\beta a_\alpha + s_\gamma a_\alpha = 0$, 从而 $\alpha = -\dfrac{s_\gamma}{r_\beta}$.

由此我们得到了 y 分母次数 α 的一个上界:

引理16.2. *若 $\gamma = \beta - 1$ 且 $-\dfrac{s_\gamma}{r_\beta}$ 为正整数, 则*

$$\alpha \leq \max\left\{\delta - \beta + 1, -\frac{s_\gamma}{r_\beta}\right\},$$

否则

$$\alpha \leq \min\{\delta - \beta + 1, \delta - \gamma\}.$$

有了引理 16.2 , 接下来便可以对 y 的分子进行待定系数法求解了, 最终可得到方程 (16.1) 在 $K = \mathbb{Q}(x)$, $\mathbb{R}(x)$ 或 $\mathbb{C}(x)$ 的解 y.

16.1.2 一般情形

上面我们讨论了 K 为最简单的有理函数域的情形. 对于一般情形, Bronstein[41] 证明了如下定理.

定理16.1 (Bronstein)**.** *若在 K 的任意有限次代数扩张中初等积分和 Risch 方程问题都可解, 设 θ 为 K 上的超越初等生成元, 则在 $K(\theta)$ 的任意有限次代数扩张中初等积分和 Risch 方程问题也都可解.*

由于 Risch[147], Rothstein[151], Davenport[63] 解决了超越初等扩张的积分问题和 Risch 方程问题, Trager[21], Davenport[62] 解决了代数扩张的积分问题和 Risch 方程问题. 从而根据定理 16.1 , 便可以从理论上得到所有初等函数的初等积分和 Risch 方程问题的解决方案了.

16.2 一阶线性微分方程

管求积分实际上就是最简单的一阶方程 $y'(x)=f(x)$, 但仅仅此问题解决起来难度已经相当大了. Risch 微分方程是最简单的一类一阶微分方程(线性方程), 当然我们求的仅仅是 K 中的解, 而非一般的 K 上的初等函数解. 虽然我们知道一阶线性方程 (16.1) 有通解表达式(参考 [11])

$$y=e^{-\int f}\cdot\left(c+\int ge^{\int f}\right),$$

但这并没有解决一般问题, 因为为了算出表达式中的指数积分, 我们还得回到解 Risch 方程问题上.

不过以下的定理 [64] 使我们能够避免循环论证而得到 K 上可能的初等函数解.

定理16.2 (Davenport)**.** *设 K 上的 Risch 方程在 K 上有初等函数解, 则有一个解 $y\in K$ 或者 $e^{\int f}$ 在 K 上代数(从而可以求出 y).*

证明. 记 $F=K(\int f)$.

1. 若 $e^{\int f}$ 在 F 上代数, 则由 Liouville 定理(定理 15.1), $\int f$ 必为 $\hat{K}$ 上对数函数的和, 从而 $e^{\int f}$ 必也在 K 上代数.
2. 若 $e^{\int f}$ 在 F 上超越, 由于 y 在 K 上初等, 自然在 F 上初等, 故 $\int ge^{\int f}=ye^{\int f}$ 也在 F 上初等. 记 $\theta=e^{\int f}$ 为 F 上的超越指数函数, 由分解引理可知 $\int g\theta=h\theta$, 其中 $h\in F$(注意指数函数求导后次数不变), 从而有解 $y=h\in F$. 若 $F=K$, 则有 $y\in K$, 否则 $\int f$ 中必有 $\hat{K}$ 上的对数函数. 可设 $y=p\big(\int f\big)$, 其中 $p(x)\in K(x)$, 而由方程 (16.1) 可得
$$p'\left(\int f\right)+p\left(\int f\right)=\frac{g}{f}\in K,$$
可知必有 $p'+p\in K$, 也有 $y\in K$.

这就完成了证明. □

因此求一阶线性方程可以通过求 K 中的解, 或是 K 上的代数函数积分来完成.

注200. 由于 $\int g\theta=h\theta$ 中积分常数的存在, 注意并没有要求所有解 y 都属于 K, 例如方程

$$y'(x)+y(x)=x+1$$

的通解为 $x+ce^{-x}$, 的确有属于 $K=\mathbb{Q}(x)$ 的解 $y(x)=x$, 但并非解所有都是.

16.3 微分 Galois 理论

接下来要讨论的自然是一般高阶线性方程如何求解的问题. 对一般的高阶线性方程

$$y^{(n)} + a_1(x)y^{(n-1)} + \cdots + a_n(x)y = f(x),$$

可化为线性方程组矩阵形式

$$\mathbf{y}' = \mathbf{A}(x)\mathbf{y} + \mathbf{f}(x),$$

其中

$$\mathbf{y} = \begin{bmatrix} y \\ y' \\ \vdots \\ y^{(n-2)} \\ y^{(n-1)} \end{bmatrix}, \mathbf{f}(x) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ f(x) \end{bmatrix},$$

$$\mathbf{A}(x) = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & & \cdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_n(x) & -a_{n-1}(x) & -a_{n-2}(x) & \cdots & -a_1(x) \end{bmatrix}.$$

设对应的齐次方程的基本解矩阵为 $\mathbf{\Phi}(x)$, 则非齐次线性方程的通解为(参考 [11])

$$\mathbf{y}(x) = \Phi(x)\left(\mathbf{c} + \int \Phi^{-1}(s)\mathbf{f}(s)\right) ds.$$

因此我们可以首先考虑齐次方程的求解. 在求解多项式方程的过程中诞生了伟大的 Galios 理论, 解线性微分方程也有一个对应物 — 微分 Galois 理论. 在这里我们做一个简要介绍, 这些理论也是 Kovacic 著名的二阶线性方程求解算法的基础(尽管最终的算法可以完全摆脱微分 Galois 理论的语言). 首先引入几个重要的概念(可参看 [170]).

定义16.1 (Picard-Vessiot 域)**.** 设 $\mathbf{y}' = \mathbf{A}\mathbf{y}$ 为 K 上的齐次线性方程组, 微分代数 $L \supseteq K$ 称为关于此方程的 Picard-Vessiot 域, 如果 $C(K)$ 为代数闭域, 且存在方程的基本解矩阵 $F \in GL_n(L)$, 使得 F 的矩阵元在 K 上生成 L.

例16.1. 考虑 $K = \mathbb{C}(x)$ 上的二阶线性微分方程 $y'' = ry$, 设 η, ζ 为方程的一个基础解系, 则一个基本解矩阵为

$$\Phi = \begin{bmatrix} \eta & \zeta \\ \eta' & \zeta' \end{bmatrix}$$

故 Picard-Vessiot 域为 $\mathbb{C}(x)(\eta,\zeta,\eta',\zeta')$.

定义16.2 (微分 Galois 群)**.** 设 $\mathbf{y}'=\mathbf{A}\mathbf{y}$ 为 K 上的齐次线性方程组, 其微分 Galois 群 $G=\mathrm{Gal}(L/K)$ 定义为其 Picard-Vessiot 域 L 上的微分 K-代数自同构群, 即 G 的元素为 L 上的所有 K-代数自同构 σ, 满足 $\sigma(f')=(\sigma(f))'$.

例16.2. (续例 16.1)设 $\sigma\in G$ 为微分 Galois 群中的元素, 则 $\sigma(\eta),\sigma(\zeta)$ 仍为方程的基础解系, 可在基 η,ζ 下表示为

$$\sigma(\eta)=a_\sigma\eta+b_\sigma\zeta,\quad \sigma(\zeta)=c_\sigma\eta+d_\sigma\zeta,$$

将其写为

$$\sigma(\eta\ \zeta)=(\eta\ \zeta)\begin{bmatrix}a_\sigma & c_\sigma\\ b_\sigma & d_\sigma\end{bmatrix}=(\eta\ \zeta)M_\sigma,$$

其中 M_σ 为 $\mathbb{C}$ 上的 2×2 矩阵, 于是 σ 可视为 $GL_2(\mathbb{C})$ 元素.

下面的结果将微分 Galois 群与代数群建立了联系, 具有根本的重要性.

定义16.3 (代数群)**.** 若 G 既是群又是仿射簇, 且群的乘法与求逆运算为仿射簇的态射, 则称 G 为代数群.

注201. 关于仿射簇(定义 13.14)及其态射, 感兴趣的读者可参看 [82] 的第一章. 直观而言, 代数群同时具备群和仿射簇结构, 并且这两种结构在某种意义下是"相容"的.

例16.3. 例如特殊线性群 $SL_n(\mathbb{C})$ 为多项式方程 $\det(x_{ij})=1$ 决定的仿射簇, 且群运算为态射, 故 $SL_2(\mathbb{C})$ 即为一个代数群. 同样的, 也可以知道特殊正交群 $SO_n(\mathbb{C})$ 也是一个代数群.

定理16.3. *微分 Galois 群 G(视为 $GL_n(C(K))$ 的子群)为代数群, 且 L 的 G-不变域为 K.*

定理的证明需要较多的准备(感兴趣的读者可参看 [170]), 因此我们在这里只给出几个例子.

例16.4. 考虑 $\mathbb{C}(x)$ 上的方程 $y''=y$, 则 Picard-Vessiot 域 $L=\mathbb{C}(x)(e^x,e^{-x})=\mathbb{C}(x,e^x)$, 且由 $(\sigma(e^x))'=\sigma((e^x)')=\sigma(e^x)$ 知

$$\sigma(e^x)=a_\sigma e^x,\quad \sigma(e^{-x})=a_\sigma^{-1}e^{-x},$$

即有

$$M_\sigma=\begin{bmatrix}a_\sigma & \\ & a_\sigma^{-1}\end{bmatrix}.$$

可知 M_σ 全体的确构成 $GL_2(\mathbb{C})$ 的代数子群.

例16.5. 考虑 $\mathbb{C}(x)$ 上的方程 $y''=-\dfrac{1}{4x^2}y$, 则 Picard-Vessiot 域 $L=\mathbb{C}(x)(\sqrt{x},\sqrt{x}\ln x)$, 且由 $(\sigma(\sqrt{x}))'=\sigma\left(\dfrac{1}{2\sqrt{x}}\right)=\dfrac{1}{2\sigma(\sqrt{x})}$ 解得

$$\sigma(\sqrt{x})=\pm\sqrt{x}.$$

又由 $(\sigma(\sqrt{x}\ln x))'=\sigma\left(\dfrac{\ln x}{2\sqrt{x}}+\dfrac{1}{\sqrt{x}}\right)=\dfrac{\sigma(\sqrt{x}\ln x)}{2x}+\dfrac{1}{\sigma(\sqrt{x})}$ 解得

$$\sigma(\sqrt{x}\ln x)=c_\sigma\sqrt{x}\pm\sqrt{x}\ln x.$$

所以方程对应的微分 Galois 群

$$G=\left\{\begin{bmatrix}1 & c\\ & 1\end{bmatrix}\middle|c\in\mathbb{C}\right\}\bigcup\left\{\begin{bmatrix}-1 & c\\ & -1\end{bmatrix}\middle|c\in\mathbb{C}\right\},$$

的确也是 $GL_2(\mathbb{C})$ 的代数子群.

例16.6. (续例 16.1)设

$$W=\begin{vmatrix}\eta & \zeta\\ \eta' & \zeta'\end{vmatrix}$$

为 Wronsky 行列式(参考 [11]), 由于 η,ζ 为方程的一个基础解系, 可知 W 为非零常数, 而

$$\sigma\begin{bmatrix}\eta & \zeta\\ \eta' & \zeta'\end{bmatrix}=\begin{bmatrix}\eta & \zeta\\ \eta' & \zeta'\end{bmatrix}\begin{bmatrix}a_\sigma & c_\sigma\\ b_\sigma & d_\sigma\end{bmatrix},$$

可知 $\sigma(W)=W\det M_\sigma$. 又 $\sigma(W)=W\neq 0$, 可知 $\det M_\sigma=1$, M_σ 全体实际上构成 $SL_2(\mathbb{C})$ 的一个代数子群.

16.4 Lie-Kolchin 定理

我们可以考虑比初等函数更广的一类函数, 应用起来也更加方便.

定义16.4 (Liouville 函数)**.** 设 K 为一个微分代数, $K(\eta)$ 为 K 的微分扩张, 满足 $C(K(\eta))=C(K)$. η 称为在 K 上的 Liouville 生成元, 若 η 满足以下条件之一

1. η 在 K 上是代数的, 即有 $f(x)\in K[x]$ 使得 $f(\eta)=0$;
2. η 在 K 上是原函数, 即有 $\eta'\in K$;
3. η 在 K 上是原函数的指数, 即有 $\dfrac{\eta'}{\eta}\in K$.

称 $K(\eta_1,\ldots,\eta_n)$ 为 K 上的一个 Liouville 函数域, 若 $\forall 1 \leq i \leq n$, η_i 为 $K(\eta_1,\ldots,\eta_{i-1})$ 上的 Liouville 生成元(约定 $K(\eta_0) = K$). K 上的任一个 Liouville 函数域中的元素称为 K 上的 Liouville 函数.

注202. 因为对数函数满足第二条, 指数函数满足第三条, 因此初等函数一定也是 Liouville 函数. 但反之则不然, 例如 $\int e^{x^2}$ 是 Liouville 函数而非初等函数.

线性微分方程的优美理论将引导我们, 把方程 Liouville 解与一般线性群 $GL_n(\mathbb{C})$ 的代数子群联系到一起. 下面的 Lie-Kolchin 定理(参见 [107])给出了线性微分方程的解是 Liouville 解的充要条件, 在理论上十分重要.

定理16.4 (Lie-Kolchin)**.** *设 G 为线性微分方程对应的微分 Galois 群, G_0 为 G 的包含单位的连通分支, 则以下条件是等价的:*

1. *线性微分方程对应的解都是 Liouville 的;*
2. *G_0 可解;*
3. *G_0 可上三角化.*

因此为了研究线性微分方程的 Liouville 解, 我们可以转向代数子群的研究. 比如说对于二阶线性方程, 我们便需要对特殊线性群 $SL_2(\mathbb{C})$ 的代数子群有所了解.

16.5 二阶线性微分方程

考虑二阶齐次线性方程

$$z'' + az' + bz = 0,$$

其中 $a, b \in K = \mathbb{C}(x)$, 通过标准的变换 $y = e^{\frac{1}{2}\int a} z$($\mathbb{C}(x)$ 中的积分总可以求出), 可化为缺项形式

$$y'' = (b - \frac{1}{4}a^2 - \frac{1}{2}a')y.$$

因此我们下面可以只考虑形如

$$y'' = ry, \tag{16.2}$$

其中 $r \in K$ 的二阶线性方程. 在这样的特殊形式下我们知道, 若 η 是一个 Liouville 解的话, 那么 $\zeta = \eta \int \frac{1}{\eta^2}$ 是与 η 线性无关的另一个解(参考 [11]), 从而方程所有的解都是 K 上的 Liouville 解.

正如上面所说的, 特殊线性群 $SL_2(\mathbb{C})$ 的代数子群对于我们很重要, 事实上有下面的定理(参见 [106]).

定理16.5 ($SL_2(\mathbb{C})$ 代数子群的分类)**.** *设 G 为 $SL_2(\mathbb{C})$ 的代数子群, 则 G 满足以下之一:*

1. *G 共轭于 $\left\{\begin{bmatrix} c & d \\ & c^{-1} \end{bmatrix} \middle| c,d \in \mathbb{C}, c \neq 0\right\}$ 的子群, 即 G 可上三角化;*
2. *G 共轭于 $\left\{\begin{bmatrix} c & \\ & c^{-1} \end{bmatrix} \middle| c \in \mathbb{C}, c \neq 0\right\} \bigcup \left\{\begin{bmatrix} & c \\ -c^{-1} & \end{bmatrix} \middle| c \in \mathbb{C}, c \neq 0\right\}$ 的子群, 且不属于第一类;*
3. *G 是前两类以外的有限群(四面体群, 八面体群, 十二面体群);*
4. *$G = SL_2(\mathbb{C})$.*

对应于代数子群的四个类型, 如下定理就不令人意外了.

定理16.6. *方程 (16.2) 的解只有四种可能性:*

1. *有一个解 $y = e^{\int \omega}$ 为 Liouville 解, 其中 $\omega \in \mathbb{C}(x)$;*
2. *有一个解 $y = e^{\int \omega}$ 为 Liouville 解, 其中 ω 为 $\mathbb{C}(x)$ 上的二次代数元, 且第一类情形不成立;*
3. *所有解 y 在 $\mathbb{C}(x)$ 上是代数的, 为 Liouville 解, 且前两类情形不成立;*
4. *无 Liouville 解.*

证明. 我们分别讨论对应于定理 16.5 的四种情形. 设 η, ζ 为方程的基础解系.

1. 可假设 G 是上三角的, 由 $\forall \sigma \in G$, $\sigma\eta = c_\sigma \eta$, 可知 $\sigma\left(\dfrac{\eta'}{\eta}\right) = \dfrac{\eta'}{\eta} \in \mathbb{C}(x)$, 令 $\omega = \dfrac{\eta'}{\eta}$, 即得 $\eta = e^{\int \omega}$.
2. 可假设 G 为 $\left\{\begin{bmatrix} c & \\ & c^{-1} \end{bmatrix} \middle| c \in \mathbb{C}, c \neq 0\right\} \bigcup \left\{\begin{bmatrix} & c \\ -c^{-1} & \end{bmatrix} \middle| c \in \mathbb{C}, c \neq 0\right\}$ 的子群, 则 $\forall \sigma \in G$ 有
$$\sigma(\eta) = c_\sigma(\eta), \quad \sigma(\zeta) = c_\sigma^{-1}\zeta$$
或者
$$\sigma(\eta) = -c_\sigma^{-1}(\zeta), \quad \sigma(\zeta) = c_\sigma \eta.$$

于是总有 $\sigma(\eta\zeta)=\pm\eta\zeta$, $\sigma((\eta\zeta)^2)=(\eta\zeta)^2$. 由定理 16.3 可知必有 $(\eta\zeta)^2\in\mathbb{C}(x)$. 设 $\phi=\dfrac{\eta'}{\eta}+\dfrac{\zeta'}{\zeta}$, 则

$$\phi=\frac{(\eta\zeta)'}{\eta\zeta}=\frac{1}{2}\cdot\frac{((\eta\zeta)^2)'}{(\eta\zeta)^2}\in\mathbb{C}(x).$$

且由

$$\left(\frac{\eta'}{\eta}\right)'=r+\left(\frac{\eta'}{\eta}\right)^2$$

可计算得

$$-\phi''-3\phi\phi'-\phi^3+4r\phi+2r'=0. \tag{16.3}$$

设 ω 为方程

$$\omega^2-\phi\omega+\frac{1}{2}\phi'+\frac{1}{2}\phi^2+r=0$$

的一个解, 对方程两端求导可得

$$(2\omega-\phi)\omega'=\phi'\omega-\frac{1}{2}\phi''-\phi\phi'+r',$$

从而

$$2(2\omega-\phi)(\omega'+\omega^2-r)=-\phi''-3\phi\phi'-\phi^3+4r\phi+2r'=0.$$

可假定 $2\omega\neq\phi$(这是第一类情形), 于是有

$$\omega'+\omega^2=r,$$

令 $y=e^{\int\omega}$ 即为原方程的解, 且 ω 为 $\mathbb{C}(x)$ 上的二次代数元.

3. 只须证扩张次数 $[L:K]$ 有限即可. 由于 G 为有限群, 可设 $G=\{\sigma_1,\ldots,\sigma_n\}$. 取 L 中的 $n+1$ 个元素 $u_1,\ldots,u_{n+1}$, 令

$$v_k=\begin{bmatrix}\sigma_1(u_k)\\ \vdots\\ \sigma_n(u_k)\end{bmatrix},$$

则必存在 $1\leq m\leq n$ 使得 $v_1,\ldots,v_m$ 线性无关, $v_1,\ldots,v_{m+1}$ 在 L 上线性相关. 设 $v_{m+1}=c_1v_1+\cdots+c_mv_m$, 则 $\sigma(v_{m+1})=\sigma(c_1)\sigma(v_1)+\cdots\sigma(c_m)\sigma(v_m)$, 而由于 $\{\sigma_1,\ldots,\sigma_n\}$ 在任一个 $\sigma\in G$ 作用下仍为 $\{\sigma_1,\ldots,\sigma_n\}$ 的一个排列, 故由 v_k 的构造知道 $\forall 1\leq i\leq m,\sigma(c_i)=c_i$, 由定理 16.3 知 $c_i\in K$, 从而任 $n+1$ 个 L 中的元素在 K 上线性相关, $[L:K]$ 自然是有限的.

4. 由于对二阶线性方程来说, 如果有一个 Liouville 解的话, 那么所有解都是 Liouville 的. 由 Lie-Kolchin 定理 16.4 可知在这种情况下 G_0 是可解的. 但 $G = GL_2(\mathbb{C})$ 对应的 $G_0 = SL_2(\mathbb{C})$ 是不可解的. 这个矛盾便证明了方程无 Liouville 解.

定理得证. □

在知道了 Liouville 解的形式之后, 对 Liouville 解的存在性的判定便有了一些依据. 下面的必要条件主要是通过对解的奇点分析得到的. 我们知道, 设 $r = \dfrac{p}{q} \in \mathbb{C}(x)$, 其中 p, q 为互素的多项式, 则 r 的极点即为 q 的零点, 且在极点 a 处的阶数等于 a 作为 q 零点的重数. r 在 ∞ 处阶数定义为 $\deg q - \deg p$. 这实际上是更一般的离散赋值的特殊情形.

定义16.5 (阶数)**.** 设 $q \in \mathbb{C}[x]\backslash\{0\}$, 定义 q 在 ∞ 处的阶数

$$\nu_\infty(q) = -\deg(q).$$

设 $p \in \mathbb{C}[x]$ 为不可约多项式, 定义 q 在 p 处的阶数

$$\nu_p(q) = \max\{n \in \mathbb{Z} : p^n \mid q\}.$$

并约定 $\nu_\infty(0) = \infty$, $\nu_p(0) = \infty$.

设 $r = \dfrac{s}{t} \in \mathbb{C}(x)$, 定义 r 的阶数 $\nu(r) = \nu(s) - \nu(t)$(其中 ν 可以是 ν_∞ 或 ν_p).

在这样的定义下, r 在 a 处的极点阶数实际上等于 $|\nu_{x-a}(r)|$.

定理16.7. *定理 16.6 的前三种情形成立的必要条件分别是:*

1. *r 的所有极点阶数为偶数或 1, 在 ∞ 处的阶数为偶数或大于 2;*
2. *r 至少有一个极点的阶数为 2 或大于 2 的奇数;*
3. *r 在极点处的阶数小于等于 2, 在 ∞ 处的阶数大于等于 2. 可设部分分式分解*

$$r = \sum_i \frac{\alpha_i}{(x-c_i)^2} + \sum_j \frac{\beta_j}{x-d_j},$$

则 $\sqrt{1+4\alpha_i} \in \mathbb{Q}$, $\sum\limits_j \beta_j = 0$. 令

$$\gamma = \sum_i \alpha_i + \sum_j \beta_j d_j,$$

有 $\sqrt{1+4\gamma} \in \mathbb{Q}$.

证明. 所有记号与定理 16.6 的证明中一致.

1. 不妨设 0 为极点, 考虑 r 和 ω 在 0 处的 Laurent 展式(其它点处类似考虑)

$$\begin{aligned} r &= \alpha x^n + \cdots, \\ \omega &= b x^m + \cdots. \end{aligned} \tag{16.4}$$

由 $\omega' + \omega^2 = r$ 可得

$$mbx^{m-1} + \cdots + b^2 x^{2m} + \cdots = \alpha x^n + \cdots.$$

从而 $n \geq \min\{m-1, 2m\}$. 故当 $n \leq -3$ 时, 必有 $m < -1$, $2m < m-1$, 知 $n = 2m$. 所以极点阶数为偶数或者 1. 再考虑作为 ∞ 处的 Laurent 展式 (16.4) , 则有 $n \leq \max\{m-1, 2m\}$. 故当 $n \geq -1$ 时, 必有 $m > -1$, $2m > m-1$, 知 $n = 2m$. 所以 ∞ 处的阶数为偶数或大于 2.

2. 由 $(\eta\zeta)^2 \in \mathbb{C}(x)$ 但 $\eta\zeta \in \mathbb{C}(x)$(否则为第一类情形), 不失一般性, 可设

$$(\eta\zeta)^2 = x^e \prod_i (x - c_i)^{e_i},$$

其中 e 为奇数. 则

$$\phi = \frac{1}{2} e x^{-1} \cdots.$$

由式 (16.3) 及展式 (16.4) 可得

$$\left(e - \frac{3}{4}e^2 + \frac{1}{8}\right) x^{-3} + \cdots = 2\alpha(e+n)x^{n-1} + \cdots.$$

而 e 为奇数, $e - \frac{3}{4}e^2 + \frac{1}{8}x^{-3} = \frac{1}{8}e(e-2)(e-4) \neq 0$, 可知 $n = -2$ 或 $n = -e, n < -2$. 所以 r 至少有一个极点阶数为 2 或为大于 2 的奇数.

3. 不妨设 0 为一个极点(其它极点类似考虑), 由于解为 $\mathbb{C}(x)$ 上的代数函数, 可设在 0 处的 Puiseux 展式

$$\eta = c x^{\mu} + \cdots,$$

其中 $\mu \in \mathbb{Q}$, 由 $\eta'' = r\eta$ 及展式 (16.4) 可得

$$\mu(\mu-1)cx^{\mu-2} + \cdots = \alpha c x^{\mu+n}.$$

可知 $n \geq -2$, 即 r 在极点处的阶数小于等于 2. 并且若 $n = -2$ 则有 $\mu(\mu-1) = \alpha$, 可知 $\sqrt{1+4\alpha} \in \mathbb{Q}$. 于是可设

$$r = \sum_i \frac{\alpha_i}{(x-c_i)^2} + \sum_j \frac{\beta_j}{x-d_j} + P,$$

其中 $P\in\mathbb{C}[x]$. 再考虑 ∞ 处 r 的 Laurent 展式(为了不混淆, 我们将 r 的系数换一个记号)

$$r=\gamma x^m+\cdots.$$

同样可得

$$\mu(\mu-1)cx^{m-2}+\cdots=\gamma cx^{\mu+m}.$$

可知 $m\leq -2$, $P=0$, 即有 r 在 ∞ 处的阶数大于等于 2. 由

$$\begin{aligned}r&=\sum_i\frac{\alpha_i}{(x-c_i)^2}+\sum_j\frac{\beta_j}{x-d_j}\\&=\sum_j\beta_jx^{-1}+\left(\sum_i\alpha_i+\sum_j\beta_jd_j\right)x^{-2}+\cdots\end{aligned}$$

可知 $\sum\limits_j\beta_j=0$, $\gamma=\sum\limits_i\alpha_i+\sum\limits_j\beta_jd_j$. 再由 $\mu(\mu-1)=\gamma$ 知 $\sqrt{1+4\gamma}\in\mathbb{Q}$.

证毕. □

例16.7. 考虑 Airy 方程

$$y''=xy,$$

则 $r=x$ 无极点(阶数均为零)且在 ∞ 处的阶数为-1, 不满足任一条必要条件, 故方程无 Liouville 解, 其一组基解称为 Airy 函数. 同样可以知道, 当 $n\geq 1$ 时, 方程 $y''=x^ny$ 都没有 Liouville 解.

下面我们来说明如何通过类似的奇点分析具体构造出所需要的 Liouville 解. 首先设 r 满足第一类必要条件, 即 r 的所有极点阶数为偶数或 1, 在 r 处的阶数为偶数或大于 2.

设解 $\eta=e^{\int\omega}$, $\omega\in\mathbb{C}(x)$. 则 ω 在 c 处的部分分式项可写为

$$\sum_{i=2}^{\nu}\frac{a_i}{(x-c)^i}+\frac{\alpha}{x-c}=:[\omega]_c+\frac{\alpha}{x-c}.$$

不妨仅考虑 0 处, 简记 $[\omega]=[\omega]_0$, 则 ω 在 0 处的 Laurent 展式可写为

$$\omega=[\omega]+\frac{\alpha}{x}+\overline{\omega},$$

其中 $\overline{\omega}$ 为 x 的非负幂次项级数. 以下分几种情况讨论.

(c_1) 设 0 处极点阶数为 1, $r=*x^{-1}+\cdots$, 由 $\omega'+\omega^2=r$ 得到

$$-\frac{\nu a_\nu}{x^{\nu+1}}+\cdots+\frac{a_\nu^2}{x^{2\nu}}=\frac{*}{x}+\cdots.$$

可知 $\nu \le 1$, $[\omega]=0$. 从而有

$$-\frac{\alpha}{x^2}+\overline{\omega}'+\frac{\alpha^2}{x^2}+\frac{2\alpha}{x}\overline{\omega}+\overline{\omega}^2=\frac{*}{x},$$

可知 $\alpha^2-\alpha=0$, 而 0 是极点, 故只能有 $\alpha=1$. 从而 ω 在 0 处的部分分式为 $\dfrac{1}{x}$. 为了统一起见, 记 $\alpha^{\pm}=1$.

(c_2) 设 0 处极点阶数为 2, $r=bx^{-2}+*x^{-1}+\cdots$, 同样的推理可以得到 $[\omega]=0$, $\alpha^2-\alpha=b$, 从而 ω 在 0 处的部分分式为 $\dfrac{\alpha^{\pm}}{x}$, 其中 $\alpha^{\pm}=\frac{1}{2}(1\pm\sqrt{1+4b})$.

(c_3) 设 0 处极点阶数为 $2\nu\ge 4$, 可设

$$[\sqrt{r}]=\frac{a}{x^{\nu}}+\cdots+\frac{*}{x^2}.$$

由 $(\sqrt{r}-\omega)(\sqrt{r}+\omega)=\omega'$ 可得 $[\omega]=\pm[\sqrt{r}]$. 设 ω 在 0 处的部分分式为 $\pm[\sqrt{r}]+\dfrac{\alpha^{\pm}}{x}$, 经过计算可得 $\alpha^{\pm}=\frac{1}{2}(\pm\frac{b}{a}+\nu)$, 其中 b 为 $r-[\sqrt{r}]^2$ 中 $\frac{1}{x^{\nu+1}}$ 的系数.

(c_4) 设 0 非极点, 与(c_1)同样的推理可得 $[\omega]=0$, $\alpha=0$ 或 $\alpha=1$. ω 在 0 处的部分分式是 0 或者 $\dfrac{1}{x}$.

综合以上四种情况得到

$$\omega=\sum_{c\in\Gamma}\left(s(c)[\sqrt{r}]_c+\frac{\alpha_c^{s(c)}}{x-c}\right)+\sum_{i=1}^{d}\frac{1}{x-d_i}+R,$$

其中 Γ 表示 r 的极点集合, $R\in\mathbb{C}[x]$, 记号 $s(c)$ 可取$+$ 或 $-$. 因此下面的任务就是确定 d, d_i 和 R 了, 为此我们还需考虑 r 在 ∞ 处 Laurent 展式

$$\omega=R+\frac{\alpha_\infty}{x}\cdots+.$$

(∞_1) 设 ∞ 处阶数为 ν 大于 2,

$$r=\frac{*}{x^{\nu}}+\frac{*}{x^{\nu+1}}+\cdots,$$

代入 $\omega'+\omega^2=r$ 可得 $R=0$, $\alpha_\infty=0$ 或 $\alpha_\infty=1$.

(∞_2) 设 ∞ 处阶数为 2, $r=bx^{-2}+*x^{-3}+\cdots$, 同样的可得 $R=0$, $\alpha_\infty^2-\alpha_\infty=b$, $\alpha_\infty^{\pm}=\frac{1}{2}(1\pm\sqrt{1+4b})$.

(∞_3) 设 ∞ 处阶数为 $-2\nu\le 0$, $[\sqrt{r}]_\infty=ax^{\nu}+\cdots$ 表示 $\sqrt{r}$ 在 ∞ 处展式的多项式部分. 类似(c_3)可得 $R=[\sqrt{r}]_\infty$, $a_\infty^{\pm}=\frac{1}{2}(\pm\frac{b}{a}-\nu)$, 其中 b 为 $r-[\sqrt{r}]_\infty^2$ 中 $x^{\nu-1}$ 的系数.

综合以上三条可得到

$$\omega=\underbrace{\sum_{c\in\Gamma}\left(s(c)[\sqrt{r}]_c+\frac{\alpha_c^{s(c)}}{x-c}\right)+s(\infty)[\sqrt{r}]_\infty}_{\theta}+\underbrace{\sum_{i=1}^{d}\frac{1}{x-d_i}}_{P'/P}.$$

再比较两边 ∞ 展式的 $\frac{1}{x}$ 的系数可得

$$d = \alpha_\infty^{s(\infty)} - \sum_{c\in\Gamma} \alpha_c^{s(c)} \in \mathbb{N}.$$

最后还需要确定 d_i, 设 $P = \prod_{i=1}^{d}(x-d_i)$, 则 ω 可写为 $\theta + \frac{P'}{P}$, 代入 $\omega' + \omega^2 = r$ 得到

$$P'' + 2\theta P' + (\theta' + \theta^2 - r)P = 0.$$

用待定系数法求出多项式 P 即可. 得解 $\eta = e^{\int \omega} = Pe^{\int \theta}$.

以上对构造第一类解的讨论, 我们可以稍作一些总结. 为了求解 $\eta = e^{\int \omega}$, 首先通过奇点分析得到 θ(即每个奇点贡献的总和), 再通过待定系数法求解关于多项式 P 的一个微分方程, 最后根据 θ 和 P 计算 ω. 在构造另外两类解时过程也大抵如此, 只不过奇点分析更复杂一些, 关于 P 的微分方程次数会较高, 而通过 θ 和 P 计算 ω 的过程也更复杂, 需要解一个代数方程. 我们不再一一详述, 具体可参见 [106], 不过为了完整起见, 这里给出 Saunders[152] 对 Kovacic 算法的一个简化版本, 不仅利于实现, 也避免了具体求出所有奇点.

算法16.2 (Kovacic 算法简化版本).

求微分方程 $y'' = ry$ 的 Liouville 解.

1. 预处理与必要条件检测.

   (a) 设 $r = \frac{s}{t}$ 为既约分式, 作无平方分解(例如算法 9.9) $t = \prod_{i=1}^{m} t_i^i$.

   (b) 构造 $L \subseteq \{1, 2, 4, 6, 12\}$ 为 ω 在 $\mathbb{C}(x)$ 上代数次数可能的取值.

   - 若 $\forall i \geq 3$ 为奇数有 $t_1 = 1$, 且 $\nu_\infty(r)$ 大于 2 或为偶数, 则 $1 \in L$;
   - 若 $t_2 \neq 1$, 或 $\exists i \geq 3$ 为奇数有 $t_i \neq 1$, 则 $2 \in L$;
   - 若 $m \leq 2$ 且 $\nu_\infty(r) \geq 2$, 则 $4, 6, 12 \in L$.

2. 构造 θ 和 d 的组成部分.

   (a) 构造固定部分

$$d_{\text{fix}} = \frac{1}{4}(\min(\nu_\infty(r), 2) - \deg t - 3\deg t_1),$$
$$\theta_{\text{fix}} = \frac{1}{4}\left(\frac{t'}{t} + \frac{3t_1'}{t_1}\right).$$

(b) 构造 2 阶极点对应项. 设 t_2 的根为 $c_1,\dots,c_{k_2}$, 设 $b_i(i=1,\dots,k_2)$ 为 r 中 $\dfrac{1}{(x-c_i)^2}$ 的系数, 令

$$d_i=\sqrt{1+4b_i},\quad \theta_i=\frac{d_i}{x-c_i}.$$

(c) 构造大于等于 3 阶极点对应项. 当 $1\in L$ 时求得 $t_4,t_6,\dots$ 的根为 $c_{k_2+1},\dots,c_k$. 设在 $c_i(i=k_2+1,\cdots,k)$ 处的阶数为 l_i, 设 $\nu_i=l/2$, a_i 为 $[\sqrt{r}]_{c_i}$ 中 $\dfrac{1}{(x-c_i)^{\nu_i}}$ 的系数, b_i 为 $r-[\sqrt{r}]_{c_i}^2$ 中 $\dfrac{1}{x^{\nu_i+1}}$ 的系数. 令

$$d_i=\frac{b_i}{a_i},\quad \theta_i=2[\sqrt{r}]_{c_i}+\frac{d_i}{x-c_i}.$$

(d) 构造 ∞ 处阶数为 2 的对应项. 若 $\nu_\infty(r)=2$, 令 b_0 为 r 中 $\dfrac{1}{x^2}$ 的系数, 令

$$d_0=\sqrt{1+4b_0},\quad \theta_0=0.$$

(e) 构造 ∞ 处阶数小于 2 的对应项. 若 $\nu_\infty(R)<2$, 设 $\nu_0=\nu_\infty(r)/2$, a_0 为 $[\sqrt{r}]_\infty$ 中 x^{ν_0} 的系数, b_0 为 $r-[\sqrt{r}]_\infty^2$ 中 x^{ν_0-1} 的系数, 令

$$d_0=\frac{b_0}{a_0},\quad \theta_0=2[\sqrt{r}]_\infty.$$

3. 构造 θ 和 d.

(a) 取 $n\in L$, 若 $n=1$ 则 $m=k$, 否则 $m=k_2$.

(b) 其中任取 $s_i\in\{-\frac{1}{2}n,-\frac{1}{2}n+1,\dots,\frac{1}{2}n\}$, 令 $S=(s_0,\dots,s_m)$. 构造

$$d_s=n\cdot d_{\text{fix}}-\sum_{i=0}^{m}s_id_i,$$

(c) 若 $d_s\in\mathbb{N}$, 则令

$$\theta_s=n\cdot\theta_{\text{fix}}-\sum_{i=0}^{m}s_i\theta_i,$$

进行第四步的求解, 否则换一组不同的 S. 若已尝试所有 S 则换一个不同的 $n\in L$, 若已尝试所有的 n, 则算法终止.

4. 求解. 记 $\theta=\theta_s$.

(a) 若 $n=1$, 求解 d_s 次首一多项式 P, 使

$$P''+2\theta P'+(\theta'+\theta^2-r)P=0.$$

输出解 $\eta=Pe^{\int\theta}$.

(b) 若 $n=2$, 求解 d_s 次首一多项式 P, 使

$$P''+3\theta P''+(3\theta^2+3\theta'-4r)P'+(\theta''+3\theta\theta'+\theta^3-4r\theta-2r')P=0.$$

令 $\phi=\theta+\dfrac{P'}{P}$, 求解方程

$$\omega^2-\phi\omega+\left(\frac{1}{2}\phi'+\frac{1}{2}\phi^2-r\right)=0.$$

输出解 $\eta=e^{\int\omega}$.

(c) 若 $n=4,6,12$, 求解 d_s 次首一多项式 P, 使

$$\begin{cases} P_n=-P, \\ P_{i-1}=-QP_i'+((n-i)Q'-Q\theta)P_i-(n-i)(i+1)Q^2rP_{i+1}, \\ P_{-1}=0. \end{cases}$$

其中 $i=1,\dots,n$, $Q=\prod\limits_{i=1}^{m}t_i$. 求解方程

$$\sum_{i=0}^{n}\frac{Q^iP_i}{(n-i)!}\omega^i=0.$$

输出解 $\eta=e^{\int\omega}$.

16.6 高阶线性微分方程的多项式解和有理解

Kovacic 算法成功地解决了二阶线性方程的 Liouville 解, 我们看到了微分 Galois 理论的巨大威力. 尽管更加困难, 使用微分 Galois 理论也的确可以用来求解高阶线性微分方程. 例如早在 1981 年 Singer[160] 就在理论上提出了一个确定性的过程来求出所有的 Liouville 解, 在此基础上还有许多发展, 但复杂度仍太高以至于无法实际应用. 因此需要考虑更快速, 便于实现的方法. 这里我们简要地介绍 Abramov, Kvansenko[16], Bronstein[42] 提出的高阶线性方程多项式解, 有理

解, 原函数指数解的方法.

16.6.1 多项式解

在本节中设 K 为一个数域, 我们 $K[x]$ 上考虑 n 阶线性方程

$$a_n y^{(n)} + \cdots + a_1 y' + a_0 y = b, \tag{16.5}$$

其中系数 $a_i, b \in K[x]$, 我们第一步的目标是求出方程 (16.5) 在 $K[x]$ 中的多项式解. 设

$$y = y_k x^k + \cdots + y_1 x + y_0, \quad y_k \neq 0$$

为一个多项式解, 代入方程 (16.5) 得到

$$\sum_{i=0}^{k} \Big(a_i(x) \cdot k(k-1)\cdots(k-i+1) \cdot y_k x^{k-i} + \cdots \Big) = b. \tag{16.6}$$

记 $m = \max\limits_{0\leq i\leq n} \{\deg a_i(x) - i\}$, 设 $\deg a_i - i$ 在 $i = i_1, \ldots, i_s$ 处取到 m. 记

$$p(r) = \sum_{j=1}^{s} \mathrm{lc}(a_{i_j}(x)) \cdot r(r-1)\cdots(r-i_j+1),$$

其中 $\mathrm{lc}(a_{i_j}(x))$ 表示 $a_{i_j}(x)$ 的首项系数. 则 $\deg p = \max\limits_{1\leq j\leq s} \{i_j\}$, 可知当 $k \geq \deg p$ 时, 式 (16.6) 中左端最高次项为

$$\sum_{j=1}^{s} \mathrm{lc}(a_{i_j}(x)) \cdot k(k-1)\cdots(k-i_j+1) \cdot y_k x^{k+m}.$$

比较式 (16.6) 两段最高项可知或者 $k+m = \deg b$, 或者 $p(k) = 0$. 从而我们得到了 y 次数的上界

$$k \leq \max\{\deg p, \deg b - m, r_{\max}\},$$

其中 $r_{\max}$ 表示 $p(r)$ 的最大正整数根.

如同我们经常做的, 在得到 y 次数上界之后便可用待定系数法求解 y 了, 最终的未定系数便为解表达式中的任意常数. 但由于无法精确求出 $r_{\max}$, 只能采取近似的估计, 而估计的 $r_{\max}$ 往往会很大, 计算复杂度就较高. 可以考虑如下的递推方法来求解关于待定系数的线性方程组更有效.

对“截断”的线性方程 $a_j y^{(j)} + \cdots + a_0 y = b (0 \leq j \leq n)$, 同样定义如上的 m 和 $p(r)$, 记为 m_j 和 $p_j(r)$(从而 $m = m_n$, $p(r) = p_n(r)$). 假设我们搜索次数不超过 k 的多项式解, 同样比较首项系数可得

$$P_{\min\{n,k\}}(k) y_k = b_{n+m_{\min\{n,k\}}}. \tag{16.7}$$

从而求出 y_k, 再作变量替换 $y(x) \leftarrow y(x) - y_k x^k$, 得到新的关于 y 的方程

$$a_n y^{(n)} + \cdots a_0 y = b - y_k(a_n(x^k)^{(n)} + \cdots + a_0(x^k)),$$

再继续求解此方程不超过 $k-1$ 次的多项式解, 如此续行即可.

当然在通过 (16.7) 计算 y_k 的过程中要考虑 $P_{\min\{n,k\}}(k) = 0$ 的情形, 若此时等式右端为零, 则判定此方程无多项式解; 若等式右端为零, 则 y_k 可视为一个未定参数, 代入变换后的新方程进行计算. 接下来当比较系数时, 若等式左端为零, 而右端含未定参数时, 便可消去右端的某个参数. 到求得 y_0 之后, 最终可以得到一个其系数中含若干个参数的多项式, 将所有系数联立为零, 可确定参数的值或者最终出现在解中的可变常数. 这最后一个联立方程的参数个数是小于等于 $2n$ 的(参见 [16]), 复杂度比非递推的待定系数法要小很多.

16.6.2 有理解

设 $y \in \overline{K}(x)$ 为有理函数解, 则从方程 (16.5) 可知 y 的每一个极点都是 a_n 的零点, 从而 $\mathrm{den}(y)$ 的每个不可约因子都是 a_n 的因子. 为了确定 $\mathrm{den}(y)$, 只需确定 a_n 的各个不可约因子在 $\mathrm{den}(y)$ 中出现的次数即可.

设 ξ 为 y 的一个极点, $y(x) = \tilde{y}(x)(x-\xi)^{-k}$. 设 a_i 在 $x-\xi$ 处的阶数为 β_i, 即 $a_i(x) = (x-\xi)^{\beta_i} h_i(x)$, $h_i(\xi) \neq 0$. 则有

$$a_i(x) = \sum_{j=\beta_i}^{\infty} \frac{a_i^{(j)}(\xi)}{j!}(x-\xi)^j,$$

代入方程 (16.5) 得到

$$\sum_{i=0}^{n} \left(\frac{a_i^{(\beta_i)}(\xi)}{\beta_i!}(x-\xi)^{\beta_i} \cdot (-k)\cdots(-k-i+1)\tilde{y}(x)(x-\xi)^{-k-i} + \cdots \right) = b. \quad (16.8)$$

和多项式解情形类似的, 我们定义 $m = \min\limits_{0\le i\le n}\{\beta_i - i\}$, 并设 $\beta_i - i$ 在 $i_1, \ldots, i_s$ 处取到 m. 记

$$p(r) = \sum_{j=1}^{s} r(r-1)\cdots(r-i_j+1)\frac{a_{i_j}^{(\beta_j)}(\xi)}{\beta_j!},$$

则当 $k \ge m$ 时, 式 (16.8) 左端 $(x-\xi)^{-k+m}$ 项的系数为 $p(-k)$, 而 $b \in K[x]$, 可知必有 $p(-k) = 0$. 于是得到

$$k \le \max\{|r_{\min}|, m\},$$

即为 $x-\xi$ 在 $\mathrm{den}(y)$ 中的次数上界, 其中 $r_{\min}$ 表示 $p(r)$ 的最小负整数根.

因此本质上和多项式解情形一样(对 ν_∞ 的分析变为了对 $\nu_{x-\xi}$ 的分析), 我们从理论上得到了一个求有理函数的方法(求得分母, 待定分子系数). 但在 $\overline{K}$ 上分解系数 a_i 总是不很方便的. 为了合并 $x-\xi$ 的乘积, 从而将所有运算都限制在系数域 $K(x)$ 中, 我们引入如下平衡分解的概念(参见 [43])就显得很自然了.

定义16.6 (平衡分解). 设 $A, B \in K[x]$, 称 A 关于 B 是平衡的, 若 $B=0$, 或者对于 A 的任意不可约因子 $P, Q \in K[x]\backslash K$, 都有 $\nu_P(B)=\nu_Q(B)$. 称 $A=A_1^{e_1}\cdots A_n^{e_n}$ 为关于 B 的平衡分解, 若 $\forall 1\le i\le n$, A_i 无平方因子, 两两互素, 且 A_i 关于 B 是平衡的.

再设 $S=\{B_1,\dots,B_m\}\subseteq K[x]$, 称 A 关于 S 是平衡的, 若 A 对于每个 $B_j\in S$ 都是平衡的. 称 $A=A_1^{e_1}\cdots A_n^{e_n}$ 为关于 S 的平衡分解, 若此分解对于每个 $B_j\in S$ 都是平衡分解.

A 的无平方分解总是关于 A 的平衡分解, 选取不同的 B, 可以得到不同于 A 的无平方分解的平衡分解. 不严格地说, 平衡分解要比无平方分解来得"细", 但比完全分解来得"粗". 我们可以只动用 K 上的 GCD 运算, 构造出 A 关于 B 的平衡分解, 为了不偏离主题太远, 算法将在稍后给出.

设 $a_n=c_1^{e_1}\cdots c_l^{e_l}$ 为关于 $S=\{a_0,a_1,\dots,a_n\}$ 的平衡分解. 考虑 $c(x)=c_i(x)(1\le i\le l)$, 设 $c(x)=(x-\xi_1)\cdots(x-\xi_k)$ 为在 $\overline{K}$ 上的完全分解. 由于 $c(x)$ 关于 S 平衡, 故对于每个 a_j, a_j 在因子 $x-\xi_1,\dots,x-\xi_k$ 处的阶数都是相等的, 公共的阶数可记为 β_j. 同样定义 $m=\min\limits_{0\le i\le n}\{\beta_i-i\}$, 并设 β_i-i 在 $i_1,\dots,i_s$ 处取到 m. 令

$$p(r,x)=\sum_{j=1}^{s} r(r-1)\cdots(r-i_j+1)\frac{a_{i_j}^{(\beta_j)}(x)}{\beta_j!}\in K[r][x],$$

为了得到 $\mathrm{den}(y)$ 中 $c(x)$ 的次数上界, 可对 $p(r,x)$ 与 $c(x)$ 施行 Euclid 算法, 求出 $r_{\min}$ 为使 $(p(r,x),c(x))=c(x)$ 的最小负整数值, 则不难知道 $\mathrm{den}(y)$ 中 $c(x)$ 的次数 k 有上界

$$k\le\max\{|r_{\min}|,m\}.$$

这样我们避开了 $\overline{K}$ 上的分解而求出了 y 的分母. 可设 $y=\dfrac{y_1}{q}$, $q=\prod\limits_{i=1}^{l}c_i^{k_i}$($k_i$ 为次数上界). 代入原方程 (16.5) 整理可得

$$\sum_{i=0}^{n}\sum_{j=0}^{n-i}\binom{i+j}{i}\frac{a_{i+j}S_j}{q^{j+1}}y_1^{(i)}=b, \tag{16.9}$$

其中

$$S_0=1,\quad S_j=S'_{j-1}q-jS_{j-1}q'\ (j\ge1).$$

接下来的问题就是求新方程 (16.9) 的多项式解 y_1, 而这是我们已经解决了的.

注203. 将方程 (16.9) 化为多项式系数的方程可以两边乘以 q^{n+1}, 但这可能导致系数很大. 为了减小方程的系数, 可以设 $h=(q,q')$, $q_1=\dfrac{q}{h}$, $q_2=\dfrac{q'}{h}$, $h_1=\dfrac{q_1h'}{h}$, 令 $T_j=\dfrac{S_j}{h^j}$(可证 $(S_j,q^{j+1})=h^j$), 则方程 (16.9) 化为

$$\sum_{i=0}^{n}\sum_{j=0}^{n-i}\binom{i+j}{i}\frac{a_{i+j}T_j}{q_1^j}y_1^{(i)}=bq,$$

其中

$$T_0=1,\quad T_j=T_{j-1}'((j-1)h_1-jq_2)+T_{j-1}q_1\ (j\geq 1).$$

这时再两边乘以 q_1^{n+1} 可有效减小系数.

16.6.3 平衡分解

我们在求高阶线性方程的有理解过程中引入了平衡分解的概念, 其实也不难构造计算平衡分解的算法. 我们将分几步走, 每步的正确性都是不难验证的.

算法16.3 (子算法 BanFacAB).

设 $A\in K[x]$ 无平方因子, 给出 A 关于 $B\in K[x]$ 的平衡分解 $\mathtt{BanFacAB}(A,B)=\{A_1,\dots,A_n\}$.

1. 若 B 为常数, 则输出 $\mathtt{BanFacAB}(A,B)=\{A\}$, 算法终止.
2. 若 B 不为常数, 计算最大公因子 $D=(A,B)$,
   - 若 $D=1$, 则输出 $\{A\}$, 算法终止;
   - 若 $D>1$, 设 $A=D\tilde{A}$, $B=D^e\tilde{B}(e\geq 1)$, 递归调用并输出 $\mathtt{BanFacAB}(A,B)=\{\tilde{A}\}\cup\mathtt{BanFacAB}(D,\tilde{B})$, 算法终止.

算法16.4 (子算法 BanFacAS).

设 $A\in K[x]$ 无平方因子, 给出 A 关于 $S\subseteq K[x]$ 的平衡分解 $\mathtt{BanFacAS}(A,S)=\{A_1,\dots,A_n\}$.

1. 若 $|S|=1$, 设 $S=\{B\}$, 输出 $\mathtt{BanFacAB}(A,B)$, 算法终止.

2. 若 $|S|>1$, 任取 $B\in S$, 递归调用计算出 $\{A_1,\dots,A_m\}=$ `BanFacAS`$(A,S\backslash\{B\})$, 输出 $\bigcup_{i=1}^{m}$ `BanFacAB`(A_i,B), 算法终止.

上面的两个算法都假定了 A 无平方因子, 因此平衡分解中因子的次数都是 1. 下面明显的定理给出一个最终版的平衡分解算法.

定理16.8 (平衡分解). *设 $A\in K[x]$, $S\subseteq K[x]$, $A=\prod_i A_i^i$ 为无平方分解, $A_i=\prod_j A_{i_j}$ 为 A_i 关于 S 的平衡分解(可通过 `BanFacAS` 计算), 则 $A=\prod_i\prod_j A_{i_j}^i$ 为 A 关于 S 的平衡分解.*

16.7　高阶线性微分方程的指数解

高阶 Liouville 解的求解要比二阶困难得多. 1992 年, Bronstein[42] 提出了求解所有指数解(严格来说是原函数的指数解, 即解 y 满足 $\frac{y'}{y}\in\overline{K}(x)$, K 为一个数域)的一组基的方法, 并且此方法也已在 Axiom 系统上实现. 此算法在 [161] 的基础上作了一些改进, 甚至不需要在 K 上分解 $a_n(x)$, 因此也可作为高阶线性微分方程的 Singer 一般算法 [160] 一个高效的子算法实现.

在本节中仅考虑高阶 n 阶齐次方程的情形, 即

$$a_ny^{(n)}+\cdots+a_1y'+a_0y=0,$$

16.7.1　Riccati 指数与 Riccati 界

设 $y=e^{\int u}$ 为方程 (16.7) 的一个指数解. 在 $n=2$ 的情形, 对方程 $y''=ry$ 我们知道 u 满足

$$u'+u^2=r,$$

这被称为原方程关联的 Riccati 方程. 对一般的方程 (16.7) , 不难求出对应的关联 Riccati 方程为

$$a_nP_n+\cdots+a_1P_1+a_0P_0=0, \tag{16.10}$$

其中

$$P_0=1,\quad P_i=P_{i-1}'+uP_{i-1}\ (i\geq 1).$$

因此求指数解的问题就相当于求解非线性的关联 Riccati 方程在 $\overline{K}(x)$ 中的解 u.

定义16.7. 设 $S=\{Q_1,\dots,Q_m\}\subseteq K[x]$, 定义 S 在 $P\in K[x]\backslash K$ 处的 Riccati 指数及 Riccati 界分别为

$$\Gamma_P(S):=\left\{\frac{\nu_P(Q_i)-\nu_P(Q_j)}{i-j}\middle|1\leq i\neq j\leq n,Q_i,Q_j\neq 0\right\}\bigcap\mathbb{N}^+,$$

$$\delta_P(S):=\begin{cases}1 & \Gamma_P(S)=\varnothing,\\ \max\Gamma_P(S) & \Gamma_P(S)\neq\varnothing.\end{cases}$$

定义 S 在 ∞ 处的 Riccati 指数和 Riccati 界分别为

$$\Gamma_\infty(S):=\left\{\frac{\deg Q_i-\deg Q_j}{i-j}\middle|1\leq i\neq j\leq n,Q_i,Q_j\neq 0\right\}\bigcap\mathbb{N}^+,$$

$$\delta_P(S):=\begin{cases}0 & \Gamma_\infty(S)=\varnothing,\\ \max\Gamma_\infty(S) & \Gamma_\infty(S)\neq\varnothing.\end{cases}$$

在这些精巧的定义下, Bronstein 证明了以下关于解 u 结构的定理(参见 [43]).

定理16.9 (Bronstein)**.** 设 $a_n=c_1^{e_1}\cdots c_l^{e_l}$ 为关于 $S=\{a_0,\dots,a_n\}$ 的平衡分解. 则有

$$u=\sum_{i=1}^{l}\frac{B_i}{c_i^{\delta_{c_i}(S)}}+\frac{Q'}{Q}+P,$$

其中 $Q,P,B_i\in\overline{K}[x]$, $(Q,a_n)=1$ 且 $\deg B_i<\deg c_i^{\delta_{c_i}(S)}$.

限于篇幅, 我们这里只能暂时省略论证的过程, 来看看有了定理 16.9 之后能做些什么. 首先, 我们可以通过计算平衡分解计算得到 $\delta_{c_i}(S)$. 接下来可以通过试探依次求出 u 中求和的每一项组成部分. 例如我们试探 α 为求和中的一项, 令 $\overline{u}=u-\alpha$, $\overline{y}=e^{\int\overline{u}}$, 则由 $y=\overline{y}e^{\int\alpha}$ 及 $D(\overline{y}e^{\int\alpha})=e^{\int\alpha}(D+\alpha)\overline{y}$($D$ 为微分算子)可得

$$a_n(D+\alpha)^n\overline{y}+\cdots+a_1(D+\alpha)\overline{y}+a_0\overline{y}=0, \tag{16.11}$$

且计算可得

$$(D+\alpha)^i=\sum_{j=0}^{i}\binom{i}{j}P_{i-j}(\alpha)D^j,$$

可得为新的关于 $\overline{y}$ 的 n 阶线性方程, 如此续行直到求出解或者全部试探完毕.

16.7.2　多项式部分

首先我们考虑求出多项式部分 P. 记 $d = \deg P$, 利用 P_i 中 x 的最高次项为 $(\mathrm{lc}(P)x)^{in}$, 记 $m_d = \max\limits_{0\le i\le n}\{id + \deg a_i \mid a_i \neq 0\}$ 且在 $i_1, \ldots, i_s$ 处取到等号, 设

$$p_d(r) = \sum_{j=1}^{s} \mathrm{lc}(a_{i_j}) r^{i_j},$$

比较方程 (16.10) 两边 x 的最高项可知 $p_d(\mathrm{lc}(P)) = 0$. 从 [161] 命题 2.3 的证明可以知道 $\deg P \le \delta_\infty(S)$, 可知 d 的取值只有有限多种, 而 p_d 的根只有有限多个, 从而 $\mathrm{lc}(P)$ 的取值至多只有有限多种可能, 依次试探并作变量替换 $\overline{u} \leftarrow u - \mathrm{lc}(P)x^d$, 如此续行, 直到求出所有可能的多项式解的集合.

16.7.3　有理部分

考虑 c_i 对应项 $\dfrac{B_i}{c_i^{\delta_{c_i}(S)}}$. 简记 $c = c_i$, $\delta = \delta_{c_i}(S)$, $R = B_i$, 作部分分式分解

$$\frac{R}{c^\delta} = \frac{R_\delta}{c^\delta} + \cdots + \frac{R_1}{c}.$$

记 $m = \max\limits_{0\le i\le n}\{i\delta - \nu_c(a_i) \mid a_i \neq 0\}$ 且 $i_1, \ldots, i_s$ 处取到等号. 令

$$p(r) = \sum_{j=1}^{s} \left(\frac{a_{i_j}}{c^{\nu_c(a_{i_j})}} \bmod c \right) \cdot S_{\delta, i_j}(r),$$

其中

$$S_{\delta,i}(r) = \begin{cases} r^i & \delta \neq 1, \\ \prod\limits_{j=0}^{i-1} (r - jc') & \delta = 1. \end{cases}$$

则比较方程 (16.10) 分母中的 c 的最高次项可知 $p(R_\delta) \bmod c = 0$. 由 $\deg R_\delta < \deg c$ 可得到一个关于 R_δ 系数的代数方程组, 得到的 R_δ 至多有有限种可能. 接下来依次求 $R_{\delta-1}, \ldots, R_1$ 并试探所有的 c_i 对应的项, 直到求出所有可能的 $\sum\limits_{i=1}^{l} \dfrac{B_i}{c_i^{\delta_{c_i}(S)}}$.

最后求解对数导数 $\dfrac{Q'}{Q}$ 了, 而这相当于对变换后的 n 阶线性方程 (16.11) 求多项式解 Q, 是我们已经解决的问题.

注204. 算法中需要求解代数方程, 因此还显得有些复杂. 不过这些代数数是表示指数解所必要的, 无法完全避免.

16.8 二阶微分方程的特殊函数解

许多特殊函数往往来源于某一类微分方程的解(二阶的尤为常见). 当一个方程没有 Liouville 解时, 我们期望能够用特殊函数来表示方程的解, 便于进一步的研究. 精确地说, 希望找到形如 $m(x)F(\xi(x))$ 的解, 其中 $m(x)$ 为 $K(x)$ 上的 Liouville 函数, $\xi(x)\in K(x)$ 为有理函数, F 则为某一个标准方程的解(例如 Bessel 方程$y''+\frac{1}{x}y+(1-\frac{\nu^2}{x^2})=0$).

例16.8. 例如二阶微分方程 $y''=(8x+1)y$ 无 Liouville 解, 但若 $\eta(x)$ 为 Airy 方程 $y''=xy$ 的一个解(例如 Airy 函数), 则 $\eta\left(\frac{1}{4}+2x\right)$ 为原方程的一个解.

我们将在下面几小节中介绍 Bronstein 在 [44] 中建立的二阶微分方程的特殊函数求解方法.

16.8.1 变量替换

下面我们考虑二阶方程 $y''=ry$, $r\in K(x)$ 的特殊函数解. 设给出特殊函数的标准方程为

$$y''+a_1y'+a_0y=0,$$

作变量替换 $y=m(x)F(\xi(x))$, 记 $M=m(x)$, $G_0=F(\xi(x))$, $G_1=F'(\xi(x))$, $A_0=a_0(\xi(x))$, $A_1=a_1(\xi(x))$, $Z=\xi(x)$, 利用 $G_0'=G_1Z'$, $G_1'=-Z'(A_0G_0+A_1G_1)$, 可得

$$\begin{aligned}
y&=MG_0\\
y'&=M'G_0+MZ'G_1\\
y''&=(M''-A_0MZ'^2)G_0+(2M'Z'+MZ''-A_1MZ'^2)G_1
\end{aligned}$$

消去 G_0, G_1 可得

$$y''-\left(2\frac{M'}{M}+\frac{Z''}{Z'}-A_1Z'\right)y'-\left(\left(\frac{M'}{M}\right)'-\frac{M'^2}{M^2}-\frac{M'Z''}{MZ}+\right.$$
$$\left.+A_1Z'\frac{M'}{M}-A_0Z'^2\right)y=0$$

若 y 满足待求解方程 $y''=ry$, 则有

$$2\frac{M'}{M}+\frac{Z''}{Z'}-A_1Z'=0, \tag{16.12}$$

及

$$\left(\frac{M'}{M}\right)'-\frac{M'^2}{M^2}-\frac{M'Z''}{MZ}+A_1Z'\frac{M'}{M}-A_0Z'^2=r$$

可解得

$$\frac{M'}{M}=\frac{1}{2}\left(A_1Z'-\frac{Z''}{Z'}\right). \tag{16.13}$$

代入式 (16.12) 可得

$$3Z''-2Z'Z'''+(A_1^2+2A_1'-4A_0)Z'^4-4rZ'^2=0. \tag{16.14}$$

16.8.2 有理函数 Z 的求解

首先注意到, 由式 (16.13) 可得

$$M=Z'^{-\frac{1}{2}}e^{\frac{1}{2}\int A_1Z'}$$

为 Liouville 函数. 因此要求得符合标准方程的解, 关键问题是如何找出合适的有理函数 Z. 寻找的主要方法还是分析函数的奇异性, 首先求出 Z 的分母 $\mathrm{den}(Z)$, 再找到其分子次数的上界, 最后通过求解代数方程求出分子的各系数.

定理16.10 (Bronstein)**.** *设 $\mathrm{den}(r)$ 的无平方分解为 $\prod\limits_i Q_i^i$. 记 $\Delta=a_1^2+2a_1'-4a_0$, $\delta=\nu_\infty(\Delta)$. 若 $\delta<2$, 则 Z 可写为 $\dfrac{P}{Q}$, 其中*

$$Q=\prod_i Q_{(2-\delta)i+2}^i\in K[x], \tag{16.15}$$

$P\in K[x]$, 满足 $\deg P\leq\deg Q+1$(当 $\delta<0$ 时不等号严格)或者 $\deg P=\deg Q+\dfrac{2-\nu_\infty(r)}{2-\delta}$.

证明. 设 p 为不可约多项式, 使得 $\nu_p(Z)<0$, 则有 $\nu_p(Z''^2)=\nu_p(Z'Z''')=2(\nu_p(Z)-2)$, $\nu_p(\Delta(Z)Z'^4)=-\delta\nu_p(Z)+4(\nu_p(Z)-1)=(4-\delta)\nu_p(Z)-4$, $\nu_p(rZ'^2)=\nu_p(r)+2(\nu_p(Z)-1)$. 从而由 $\delta<2$ 及式 (16.14) 可知

$$(4-\delta)\nu_p(Z)-4=\nu_p(r)+2(\nu_p(Z)-1),$$

即

$$\nu_p(Z)=\frac{\nu_p(r)+2}{2-\delta}.$$

注意到 $-\nu_p(Z)$, $-\nu_p(r)$ 为分别为 $\mathrm{den}(Z)$, $\mathrm{den}(r)$ 中 p 的重数, 可知式 (16.15) 成立.

设 $\deg P>\deg Q+1$, 下证必有 $\deg P=\deg Q+\dfrac{2-\nu_\infty(r)}{2-\delta}$. 由于 $\nu_\infty(Z)\leq-2$, 可得 $\nu_\infty(Z''^2)=2\nu_\infty(Z)+4$, $\nu_\infty(Z'Z''')=2\nu_\infty(Z)+4$(当 $\nu(Z)<-2$)或

≥ -1(当 $\nu(Z)=-2$ 时), $\nu_\infty(\Delta(Z)Z'^4)=(4-\delta)\nu_\infty(Z)+4$, $\nu_\infty(rZ'^2)=\nu_\infty(r)+2(\nu_\infty(Z)+1)$, 同样由 $\delta<2$ 及式 (16.14) 可得

$$(4-\delta)\nu_\infty(Z)+4=\nu_\infty(r)+2(\nu_\infty(Z)+1),$$

即

$$\nu_\infty(Z)=\frac{\nu_\infty(r)-2}{2-\delta}.$$

再注意到 $\nu_\infty(Z)=\deg Q-\deg P$ 可得所要证的.

最后注意当 $\delta<0$ 时有 $\nu_\infty(r)=\delta$, 从而 $\nu_\infty(Z)=1$, 故不等号成立与后一种情形是重合的, 这就完成了证明. □

由此我们得到了想要的 Q 的表达式和 P 次数的上界, 将 P 的系数待定代入 (16.14) 可以得到一个代数方程组, 最后解此方程组即可. 另外, 为保证得到的 $Z=\dfrac{P}{Q}$ 不是常数(注意常数总是解), 我们可以添加额外的一些限制方程. 设 $P=\sum\limits_{j=0}^{n}c_jx^j$, $Q=\sum\limits_{i=0}^{n}q_ix^i$, , $\forall 0\leq N\leq n, q_N\neq 0$, 加入方程 $\sum\limits_{j=0}^{N}(q_jc_N-q_Nc_j)w_j=1$, 其中 w_j 为新的待定元. 可以看出, 解新的方程组可以保证至少有一个 j 使得 $q_jc_N\neq q_Nc_j$, 从而 Z 非常数.

函数 a_0, a_1 也允许带上参数(例如 Bessel 方程中的参数 ν), 将它们也视作方程组中的未定元, 求解代数方程组时可以将参数确定下来.

注205. 用此方法也可求解一些特殊形式的代数解, 例如形如 $Z=P(x^{\frac{1}{2-\delta}})\prod\limits_i Q_i^{\frac{i-2}{2-\delta}}$ 的解.

16.8.3 经典特殊函数

在本节的最后我们给出定理 16.10 在一些经典特殊函数的情形, 便于实际应用.

- Airy 函数, 由方程

  $$y''=xy$$

  给出. $\Delta=4x$, $\delta=-1<0$, 可知 $Q=\prod\limits_i Q_{3i+2}^i$, $\deg P\leq\deg Q$ 或 $\deg P=\deg Q+\dfrac{2-\nu_\infty(r)}{3}$, $M=\sqrt{\dfrac{1}{Z'}}$.

- Bessel 函数, 由方程

  $$y''+\frac{1}{x}y'+\left(\varepsilon-\frac{\nu^2}{x^2}\right)y=0$$

给出, 其中 $\varepsilon = \pm 1$. $\Delta = \dfrac{4\nu^2-1}{x^2} - 4\varepsilon$, $\delta = 0 < 2$, 可知 $Q = \prod\limits_i Q_{2i+2}^i$, $\deg P \leq \deg Q + 1$ 或 $\deg P = \deg Q + 1 - \dfrac{\nu_\infty(r)}{2}$, $M = \sqrt{\dfrac{Z}{Z'}}$.

- Kummer 函数, 由方程

$$y'' + \left(\frac{\nu}{x} - 1\right) y' - \frac{\mu}{x} y = 0$$

给出. $\Delta = \mu + \dfrac{4\mu-\nu}{x} + \dfrac{\nu^2-2n}{x^2}$, $\delta = 0 < 2$, 可知 $Q = \prod\limits_i Q_{2i+2}^i$, $\deg P \leq \deg Q + 1$ 或 $\deg P = \deg Q + 1 - \dfrac{\nu_\infty(r)}{2}$, $M = e^{-\frac{1}{2}\int Z} \sqrt{\dfrac{Z^\nu}{Z'}}$.

- Whittaker 函数, 由方程

$$y'' = \left(\frac{1}{4} - \frac{\mu}{x} - \frac{1/4-\nu^2}{x^2}\right) y$$

给出, $\Delta = 1 - \dfrac{4\mu}{x^2} - \dfrac{1-4\nu^2}{x^2}$, $\delta = 0 < 2$, 可知 $Q = \prod\limits_i Q_{2i+2}^i$, $\deg P \leq \deg Q + 1$ 或 $\deg P = \deg Q + 1 - \dfrac{\nu_\infty(r)}{2}$, $M = \sqrt{\dfrac{1}{Z'}}$.

A maTHμ 系统简介

maTHμ 项目组由来自清华大学数学, 物理, 计算机, 电子, 自动化等不同院系的学子所组成, 在清华大学团委科创中心及清华大学学生科协的支持和指导下, 致力于开发具有自主知识产权的计算机代数系统 maTHμ[1]. 自 2007 年 9 月学生立项以来, maTHμ 项目已经持续运行了近两年时间, 设计并开发了 maTHμ 1.0 版本的内核与用户图形界面, 部分功能已经可以通过网络计算平台测试版[2]进行免费访问. 本附录将会对 maTHμ 系统的主要特点与功能进行一个概要性的介绍.

A.1 系统架构与特点

系统架构设计

maTHμ 的整体系统架构设计如图 A.1 所示. 图中内层箭头表示了数据流在系统内部各模块中传递的过程, 而外层箭头则展示了当用户在图形界面中中输入一个的积分命令

$$\int x^5\sqrt{ax+b}\,dx$$

之后, 数据流在各个模块中的具体表现形式. 用户的输入命令在图形前端中被储存为适宜使用类 TEX 算法绘制的盒子模型对象, 输入命令线性化后通过通信层传递给解释器. 解释器通过词法分析及语法分析将命令解析为表达式树, 然后由求值器进行逐层求值, 将计算结果再通过通信层传回图形前端显示为图形公式. 求

[1] http://www.mathmu.cn

[2] http://www.mathmu.cn/Platform.html

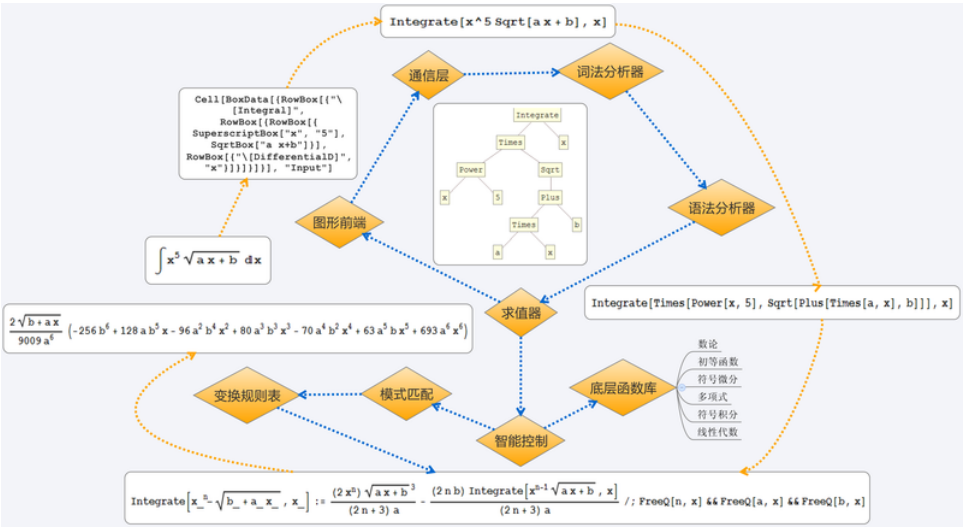

图 A.1: maTHμ 系统架构

值模型主要分为底层函数库和模式匹配两部分, 两者由智能控制模块进行控制. 图形前端则采用 MVC(Model-View-Controller)架构, 能够将任意对象(如字符, 图形, 控件等)作为数学表达式参与运算.

可移植性

计算机代数系统结构复杂, 功能繁多, maTHμ 系统遵循着各层分离, 统一接口的设计思想. 举例来说, 所有底层数据结构都来自`var`, 使得无论底层函数的实现如何变动, 例如将所有 GMP 导出函数换为 NTL 导出函数, 所有函数接口都仍可以保持不变而不影响上一层符号框架及解释器. 正是因为如此, 系统各层子系统内部都有足够的灵活性, 同一个图形前端可以与不同类型的解释器交互, 除了通信部分以外无需改动.

同样的, 同一个解释器也可以与不同实现的底层函数库进行交互. 通过更换词法/语法分析器, 还可以方便地支持不同语法的输入而无需改动计算内核. maTHμ 目前已经能够接收类 Mathematica 语法与类 Lisp 语法等不同语言的输入, 未来还可以更广泛的支持 Maple, Matlab 等通用计算语言. 目前 maTHμ 的词法定义通过手工编码的有限状态自动机(DFA)来描述, 具体实现时对忽略空格, 删除嵌套注释, 识别 Unicode 转义字符以及识别字符串中 C 风格的转义字符等功能做了特殊处理. 文法定义可以通过手工编写的 LL(1)递归下降分析框架来描述, 为了简洁起见, 具体实现时对表达式主体采用算符优先分析, 原子表达式以及大量上下文相关的特殊文法则采用手工处理. 目前支持 60 余种算符的类 Mathematica 语法可使程

序的编写更加接近于数学思维.

可扩展性

maTHμ 的内核可支持动态链接库和程序包等多种类型的模块扩展, 这样可以使内核在保持足够轻便的同时拥有可扩展的计算能力. 从开发者的角度来看, 扩展模块可以独立编译, 大大缩短了调试周期, 使得二次开发灵活方便. 从用户的角度来看, 同一个扩展功能, 既可以由 maTHμ 语言的程序包提供, 又可以由动态链接库等二进制模块提供以获得更高的效率, 还可以设定启动配置, 使得扩展模块在启动时自动加载, 这样就完全屏蔽了内核和扩展模块的差别. 事实上, 除了词法/语法分析器, 求值器, 模式匹配器和一些最基本的函数之外, maTHμ 绝大部分功能都由扩展模块提供.

A.2 基本功能

maTHμ 包含了高精度运算, 数论, 线性代数, 初等函数, 多项式, 方程求解, 符号微积分, 函数图形, 数据可视化等等的计算机代数系统的核心功能, 并且支持过程式, 规则式, 函数式等多模式编程. maTHμ 用户界面如图 A.3所示.

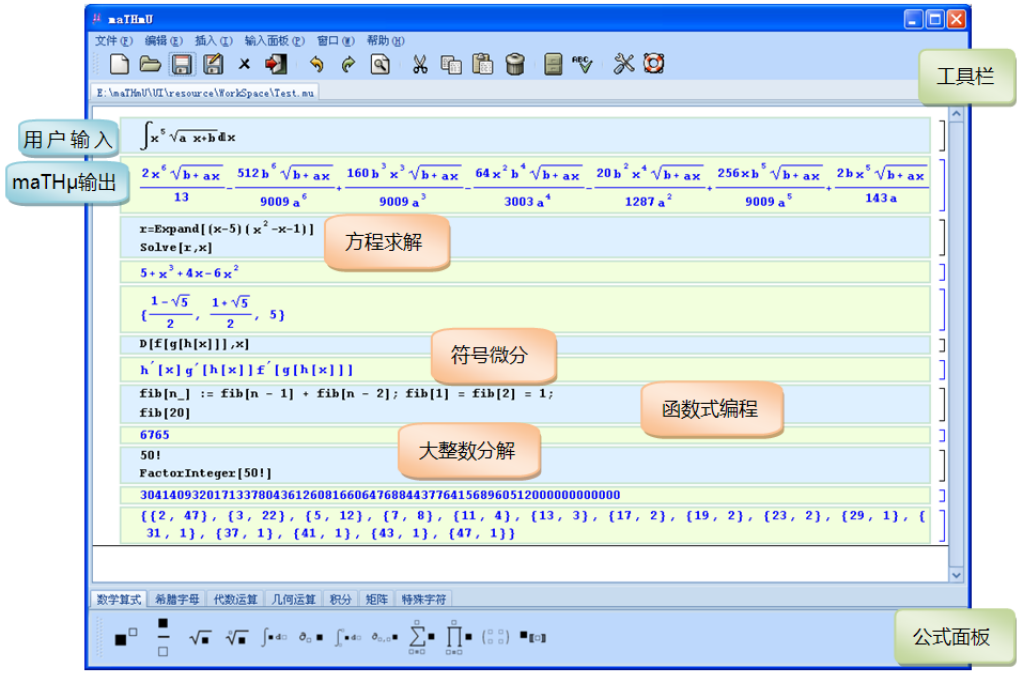

图 A.2: 系统用户界面

基本运算与函数

maTHμ 能高效地进行任意精度整数, 实数运算, 支持大量数论函数, 组合函数, 初等函数和数学常数的任意精度计算, 符号表达式展开, 合并同类项, 表达式化简等. 限于篇幅, 我们仅在此举几个简单的例子:

```
Input : 40! (*阶乘*)
Output: 815915283247897734345611269596115894272000000000
Input : FactorInteger[2^(2^6) + 1] (*整数因子分解*)
Output: {{274177, 1}, {67280421310721, 1}}
Input : Expand[(x + 2y)^4] (*表达式展开*)
Output: 81 + 216x + 216x^2 + 96x^3 + 16x^4
Input : N[π, 100] (*求圆周率至 100 位精度*)
Output: 3.14159265358979323846264338327950288419716939937
          51058209749445923078164062862089986280348253421
          17068
```

多项式与方程

功能包括多项式的最大公因子, 因子分解, 复合分解, Gröbner 基, 特征列, 多项式方程(组)求解等等. 示例如下:

```
Input : Factor[x^20 - 1] (*多项式因子分解*)
Output: (1 + x)(1 + x^2)(-1 + x)(1 + x + x^2 + x^3 + x^4)
          (1 - x + x^2 - x^3 + x^4)(1 - x^2 + x^4 - x^6 + x^8)
Input : GroebnerBasis[{x^2 - 2y^2, xy - 4}, {x, y}] (*Gröbner 基*)
Output: {-8 + y^4, -y^3 + 2x}
Input : Solve[25 + 35x + 11x^2 + x^3] (*方程求解*)
Output: {{x → -1}, {x → -5}, {x → -5}}
```

符号微积分

功能包括求导数(包括高阶导数和偏导数等), 不定积分和定积分, 将函数展成幂级数, 求函数极限等.

```
Input : D[f[g[h[x]]], x] (*函数微分*)
Output: f'[g[h[x]]] g'[h[x]] h'[x]
```

```
Input : ∫(Sin[3x+2]-1/√(x^2+1))dx (*函数积分*)
Output: -ArcSinh[x]-1/3Cos[2+3x]
```

多模式编程

支持过程式, 规则式, 函数式等多模式编程, 示例如下:

```
Input : 3^10 (*直接计算*)
Output: 59049
Input : pow[x_, m_] := x pow[x, m - 1]; pow[x_, 0] := 1;
        pow[3, 10] (*规则式编程*)
Output: 59049
Input : Clear[pow];
        pow[x_, m_] := Module[{r = x, t = x, n = m},
          If[n <= 0, Return[1]]; --n;
          While[n > 0, If[OddQ[n], r *= t]; If[n > 1, t *= t];
           n = IntegerPart[n/2]]; r];
        pow[3, 10] (*过程式编程*)
Output: 59049
```

利用 maTHμ 提供的符合数学思维的编程方式, 我们可以非常方便进行符号计算. 例如以下一段从符号微分程序包中摘取的代码片断, 便完全定义好了符号微分的数学规则:

```
(* 常数 *)
$D[a_, x_] := 0 /; FreeQ[a, x];
$D[a_ f_, x_] := a $D[f, x] /; FreeQ[a, x] && ! FreeQ[f, x];
(* 线性 *)
$D[f_ + g_, x_] := $D[f, x] + $D[g, x];
(* 乘法规则 *)
$D[f_ g_, x_] := $D[f, x] g + f $D[g, x] /; ! FreeQ[f, x]
  && ! FreeQ[g, x];
(* 链式法则 *)
$D[f_[g_], x_] := (g /. $DC[f]) $D[g, x] /;
  MemberQ[$D$headlist, f] && g =!= x;
$D[f_[g_], x_] := Derivative[1][f][g] $D[g, x] /;
  ! MemberQ[$D$headlist, f];
```

A.3　网络计算平台

maTHμ 内核支持文件, 共享内存, 管道和网络套接字等多种进程间通信方式, 并以输入输出流的方式将它们抽象出来, 可以采用统一的通信层接口来调用用这些内部原理极不相同的通信方式. 对多种进程间通信方式的支持大大降低了系统的耦合度, 例如解释器内核是用 C++ 编写, 而用户图形前端使用 Java编写, 不过两者的不同平台并没有阻碍前端与内核的通信. 内核中还内建了对多线程运算的完整支持, 可以基于这些特点搭建由运行着 maTHμ 的机群组成的大规模并行计算环境.

在如上的设计之下, maTHμ 项目组开发了基于 maTHμ 内核的网络计算平台, 用户可以通过 http://www.mathmu.cn/Platform.html 访问该平台的测试版(如图 A.3). 用户无需安装任何软件, 只需一个标准的网络浏览器即可方便地享受 maTHμ 内核的计算功能,

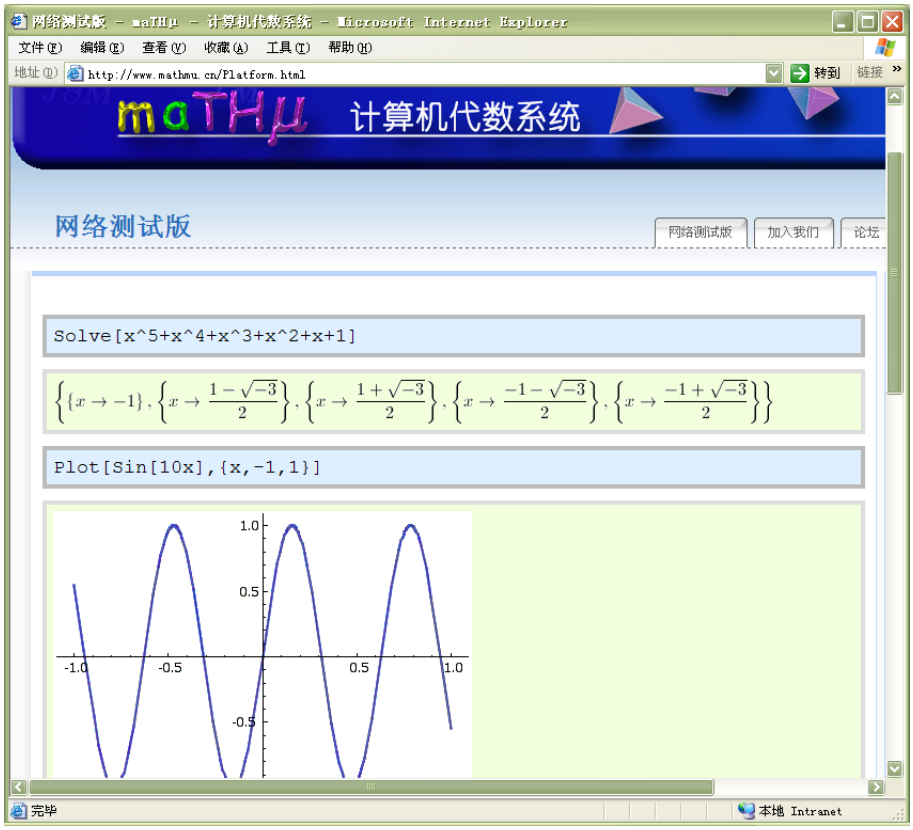

图 A.3: 网络测试版

参考文献

[1] 王竹溪，郭敦仁, 特殊函数概论, 北京大学出版社, 北京, 1973.

[2] 数学手册编写组, 数学手册（第1版）, 高等教育出版社, 北京, 1979.

[3] 吴文俊, 几何定理机器证明的基本原理（初等几何部分）, 科学出版社, 北京, 1984.

[4] 冯克勤译，聂灵沼校, 代数学, 湖南教育出版社, 长沙, 1985.

[5] 徐献瑜, *Pade*逼近概论, 上海科学技术出版社, 上海, 1990.

[6] 聂灵沼，丁石孙, 代数学引论（第2版）, 高等教育出版社, 北京, 2000.

[7] 李庆扬，王能超，易大义, 数值分析, 清华大学出版社, 北京, 2001.

[8] 裴定一，祝跃飞, 算法数论, 中国科学院研究生教学丛书, 科学出版社, 北京, 2002.

[9] 周梦, 计算代数与应用, 武汉大学出版社, 武汉, 2002.

[10] 冯克勤, 初等数论及应用, 北京师范大学出版社, 北京, 2003.

[11] 丁同仁，李承治, 常微分方程教程, 高等教育出版社, 北京, 7 2004.

[12] 张贤科，许甫华, 高等代数学（第2版）, 清华大学出版社, 北京, 7 2004.

[13] 张树功，雷娜，刘停战, 计算机代数基础—代数与符号计算的基本原理, 科学出版社, 北京, 2005.

[14] 王东明，夏壁灿，李子明, 计算机代数（第2版）, 清华大学出版社, 北京, 2007.

[15] 颜松远, 计算数论, 清华大学出版社, 北京, 2 ed., 2008.

[16] S. A. Abramov and K. Y. Kvansenko, *Fast algorithms to search for the rational solutions of linear differential equations with polynomial coefficients*, Proceedings of the 1991 international symposium on Symbolic and algebraic computation, (1991), pp. 267 – 270.

[17] W. W. Adams and P. Loustaunau, *An Introduction to Gröbner Bases*, vol. 3 of Graduate Studies in Mathematics, American Mathematical Society, 1994.

[18] L. M. Adleman, C. Pomerance, and R. S. Rumely, *On distinguishing prime numbers from composite numbers*, The Annals of Mathematics, 117 (1983), pp. 173–206.

[19] M. Agrawal, N. Kayal, and N. Saxena, *Primes is in p*, Annals of Mathematics, 160 (2004), pp. 781–793.

[20] A. O. L. Atkin and F. Morain, *Elliptic curves and primality proving*, Mathematics of Computation, 61 (1993), pp. 29 – 68.

[21] T. B, *Integration of algebraic functions*, Ph.D thesis, Dpt. of EECS, Massachusetts Institute of Technology, (1984).

[22] D. H. Bailey, P. B. Borwein, and S. Plouffe, *On the computation of the n'th decimal digit of various transcendental numbers*, Mathematics of Computation, (1996), pp. 1–2.

[23] R. Baillie and S. S. Wagstaff, Jr., *Lucas pseudoprimes*, Mathematics of Computation, 35 (1980), pp. 1391 – 1417.

[24] E. H. Bareiss, *Sylvester's identity and multistep integers preserving gaussian elimination*, Mathematics of Computation, 22 (1968), pp. 565–578.

[25] M. Beeler, R. W. Gosper, and R. Schroeppel, *HAKMEM*, MIT Artificial Intelligence Laboratory, Memo AIM-239, Cambridge, MA, 1972. http://www.inwap.com/pdp10/hbaker/hakmem/pi.html#item140.

[26] F. Bellard, *A new formula to compute the n'th binary digit of pi*, Bellard's Website, (1997), pp. 1–2.

[27] E. R. Berlekamp, *Factoring polynomials over finite fields*, Bell System Technical Journal, 46 (1967), pp. 1853–1859.

[28] ———, *Factoring polynomials over large finite fields*, Mathematics of Computation, 24 (1970), pp. 713–735.

[29] B. C. Berndt, *Ramanujan's Notebooks, Part IV*, Springer-Verlag, New York, 1994.

[30] L. Bertrand, *On the implementation of a new algorithm for the computation of hyperelliptic integrals*, in Proceedings of the international symposium on Symbolic and algebraic computation, International Conference on Symbolic and Algebraic Computation, New York, 1994, ACM, pp. 211 – 215.

[31] ———, *Computing a hyperelliptic integral using arithmetic in the jacobian of the curve*, Applicable Algebra in Engineering, Communication and Computing, 6 (1995), pp. 275–298.

[32] D. Bini and V. Pan, *Polynomial and matrix computations*, vol. 1, Birkhäuser, Boston, 1994.

[33] J. M. Borwein and P. B. Borwein, *Pi and the AGM*, Addison-Wesley Longman Publishing Co., Inc., 1987.

[34] ———, *More ramanujan-type series for 1/pi*, in Ramanujan Revisited: Proceedings of the Centenary Conference, University of Illinois at Urbana-Champaign, June 1-5, 1987, G. E. Andrews, B. C. Berndt, and R. A. Rankin, eds., New York, 1988, Academic Press, pp. 359–374.

[35] J. M. Borwein, P. B. Borwein, and D. H. Bailey, *Ramanujan, modular equations, and approximations to pi, or how to compute one billion digits of pi*, Amer. Math. Monthly 96, (1989), pp. 201–219.

[36] J. Bos and M. Coster, *Addition chain heuristics*, in Advances in Cryptology - Proceedings of Crypto '89, vol. 435, Springer-Verlag, 1990, pp. 400 – 407.

[37] R. J. Bradford and J. H. Davenport, *Effective tests for cyclotomic polynomials*, in Symbolic and Algebraic Computation, vol. 358 of Lecture Notes in Computer Science, Springer Berlin/ Heidelberg, 1989, pp. 244–251.

[38] R. P. Brent, *Multiple-precision zero-finding methods and the complexity of elementary function evaluation*, Analytic Computational Complexity, (1975), pp. 151–176.

[39] R. P. Brent, *An improved monte carlo factorization algorithm*, BIT Numerical Mathematics, 20 (1980), pp. 176–194.

[40] R. P. Brent and H. T. Kung, *Fast algorithms for manipulating formal power series*, J. Assoc. Comput. Mach., 25 (1978), pp. 581–595.

[41] M. Bronstein, *Integration of elementary functions*, Journal of Symbolic Computation, 9 (1990), pp. 117 – 173.

[42] ——, *Linear ordinary differential equations: breaking through the order 2 barrier*, Papers from the international symposium on Symbolic and algebraic computation, (1992), pp. 42 – 48.

[43] ——, *On solutions of linear ordinary differential equations in their coefficient field*, Journal of Symbolic Computation, 13 (1992), pp. 413 – 439.

[44] ——, *Solutions of linear ordinary differential equations in terms of special functions*, Proceedings of the 2002 international symposium on Symbolic and algebraic computation, (2002), pp. 23–28.

[45] ——, *Symbolic Integration I: Transcendental Functions(2nd ed.)*, Springer Verlag, 2005.

[46] W. S. Brown, *On euclid's algorithm and the computation of polynomial greatest common divisors*, Journal of the ACM, 18 (1971), pp. 478–504.

[47] J. W. Bruce, *A really trivial proof of the lucas-lehmer test*, The American Mathematical Monthly, 100 (1993), pp. 370–371.

[48] P. Bürgisser, M. Clausen, and M. A. Shokrollahi, *Algebraic complexity theory*, Springer Verlag, 1997.

[49] C. Burnikel, J. Ziegler, I. Stadtwald, and D. Saarbrucken, *Fast recursive division*, October 1998. Research Report.

[50] D. G. Cantor and H. Zassenhaus, *A new algorithm for factoring polynomials over finite fields*, Mathematics of Computation, 36 (1981), pp. 587–592.

[51] L. Chen et al., *Efficient matrix preconditioners for black box linear algebra*, linear algebra and its applications, 343-344 (2002), pp. 119–146.

[52] G. W. Cherry, *Integration in finite terms with special functions: the error function*, Journal of Symbolic Computation, 1 (1985), pp. 283 – 302.

[53] ——, *Integration in finite terms with special functions: the logarithmic integral*, SIAM Journal on Computing, 15 (1986), pp. 1 – 21.

[54] D. V. Chudnovsky and G. V. Chudnovsky, *Approximations and complex multiplication according to ramanujan*, in Ramanujan Revisited: Proceedings of the Centenary Conference, University of Illinois at Urbana-Champaign, June 1-5, 1987, G. E. Andrews, B. C. Berndt, and R. A. Rankin, eds., Boston, MA, 1987, Academic Press, pp. 375–472.

[55] ———, *Computer algebra in the service of mathematical physics and number theory*, Computers and Mathematics, 09 (1990), p. 232.

[56] H. Cohen, *A Course in Computational Algebraic Number Theory*, GTM 138, Springer Verlag, 1993.

[57] H. Cohen and H. W. Lenstra, Jr., *Primality testing and jacobi sums*, Mathematics of Computation, 42 (1984), pp. 297–330.

[58] D. Coppersmith, *Solving homogeneous linear equations over gf(2) via block wiedemann algorithm*, Math. Comp., 62 (1994), pp. 333–350.

[59] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, in proceedings of the nineteenth annual acm symposium on theory of computing, 1987.

[60] H. Corman著，潘金贵等译, 算法导论*(原书第2版)*, 机械工业出版社, 2006.

[61] J. S. Coron, *Finding small roots of bivariate integer polynomial equations revisited*, in Lecture Notes in Computer Science, vol. 3027, 2004, pp. 492–505.

[62] J. H. Davenport, *Intégration algorithmique des fonctions élémentairement transcendantes sur une courbe algébrique*, Annales de l'institut Fourier, 34 (1984), pp. 271–276.

[63] ———, *The risch differential equation problem*, SIAM Journal on Computing, 15 (1986), pp. 903 – 918.

[64] J. H. Davenport, Y. Siret, and E. Tournier, *Computer algebra: systems and algorithms for algebraic computation*, Academic Press, London, UK, 1988.

[65] M. Deléglise and J. Rivat, *Computing $\pi(x)$: The meissel, lehmer, lagarias, miller, odlyzko method*, Mathematics of Computation, 65 (1996), pp. 235–245.

[66] ———, *Computing the summation of the möbius function*, Experimental Mathematics, 5 (1996), pp. 291–295.

[67] J. Demmel et al., *Fast matrix multiplication is stable*, Numerische Mathematik, 106 (2007), pp. 199–224.

[68] J. D. Dixon, *Asymptotically fast factorization of integers*, Mathematics of Computation, 36 (1981), pp. 255–260.

[69] ———, *Exact solution of linear equations using p-adic expansions*, Numer. Math., 40 (1982), pp. 137–141.

[70] J. Dumas et al., *Linbox: a generic library for exact linear algebra*, in ICMS'02, Beijing, China, 2002, World Scientific, pp. 40–50.

[71] W. Eberly and E. Kaltofen, *On randomized lanczos algorithms*, in ISSAC'97, Maui, USA, 1997, ACM Press, pp. 176–183.

[72] C. M. Fiduccia, *Polynomial evaluation via the division algorithm: the fast fourier transform revisited*, in In Proceedings of the Fourth Annual ACM Symposium on the Theory of Computing, Denver CO, ACM Press, 1972, pp. 88–93.

[73] E. Frank, *On continued fraction expansions for binomial quadratic surds. iii*, Numerische Mathematik, 5 (1963), pp. 113–117.

[74] K. Geddes, S. Czapor, and G. Labahn, *Algorithms for Computer Algebra*, Kluwer Academic Publishers, 1992.

[75] S. Goldwasser and J. Kilian, *Almost all primes can be quickly certified*, in Proceedings of the eighteenth annual ACM symposium on Theory of computing, 1986, pp. 316 – 329.

[76] G. H. Golub and C. F. van Loan著，袁亚湘等译, 矩阵计算, 科学出版社, 2001.

[77] D. M. Gordon, *A survey of fast exponentiation methods*, Journal of Algorithms, 27 (1998), pp. 129 – 146.

[78] X. Gourdon, *Computation of pi(x) : improvements to the meissel, lehmer, lagarias, miller, odlyzko, deleglise and rivat method*, 2 2001. http://numbers.computation.free.fr/Constants/Primes/Pix/piNalgorithm.ps.

[79] J. E. Gower and S. S. Wagstaff, Jr., *Square form factorization*, Mathematics of computation, 77 (2008), pp. 551–588.

[80] J. Grabmeier, E. Kaltofen, and V. Weispfenning, eds., *Computer algebra handbook*, Springer Verlag, 2003.

[81] C. Gutierrez and M. Monsalve, *Fast inverse for big numbers: Picarte's iteration*, October 2005. Technical Report.

[82] R. Hartshorne著，冯克勤，刘木兰，胥鸣伟译, 代数几何, 科学出版社, 北京, 2001.

[83] K. Hensel, *Eine neue theorie der algebraischen zablen*, Mathematicsche Zeitschift, 2 (1918), pp. 433–452.

[84] M. J. Hinek and D. R. Stinson, *An inequality about factors of multivariate polynomials*, 2006. http://www.cacr.math.uwaterloo.ca/techreports/2006/cacr2006-15.pdf.

[85] H. J. Nussbaumer, *Fast Fourier Transform and Convolution Algorithms*, Springer-Verlag, Berlin, 1982.

[86] H. J. Nussbaumer著，胡光锐译, 快速付里叶变换和卷积算法, 上海科学技术文献出版社, 上海, 1984.

[87] S. M. Hong, S. Y. Oh, and H. S. Yoon, *New modular multiplication algorithms for fast modular exponentiation*, in Advances in Cryptology—Proceedings of Eurocrypt '96, vol. 1070 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 1996, pp. 166 – 177.

[88] E. Horowitz, *A fast method for interpoloation using preconditioning*, Infomation Processing Letters 1, (1972), pp. 157–163.

[89] T. W. Hungerford, *Algebra*, Springer Verlag, Berlin and New York, 1974.

[90] C. L. Hwang, *More machin-type identities.*, Math. Gaz., 81 (1997), pp. 120–121.

[91] T. Jebelean, *A generalization of the binary gcd algorithm*, Proceedings of the 1993 international symposium on Symbolic and algebraic computation, (1993), pp. 111–116.

[92] M. A. Jenkins and J. F. Traub, *A three-stage algorithm for real polynomials using quadratic iteration*, SIAM Journal on Numerical Analysis, 7 (1970), pp. 545–566.

[93] ———, *A three-stage variable-shift iteration for polynomial zeros in relation to generalized rayleigh iteration*, Numer. Math., 14 (1970), pp. 252–263.

[94] H. G. Kahrimanian, *Analytical differentiation by a digital computer*, master's thesis, Temple University, 5 1953.

[95] E. Kaltofen, *Analysis of coppersmith's block wiedemann algorithm for the parallel solution of sparse linear systems*, Mathematics of Computation, 64 (1995), pp. 777–806.

[96] E. Kaltofen and B. Saunders, *On wiedemann's method of solving sparse linear systems*, in Proc. AAECC-9, LNCS 539, Springer-Verlag, 1991, pp. 29–38.

[97] E. Kaltofen and V. Shoup, *Subquadratic-time factoring of polynomials over finite fields*, Mathematics of Computation, 67 (1998), pp. 1179–1197.

[98] Y. Kanada, *Current published world record of pi calculation*, October 2005. http://www.super-computing.org/pi_current.html.

[99] A. Karatsuba and P. Ofman, *Multiplication of many-digital numbers by automatic computers*, Proceedings of the USSR Academy of Sciences, (1962), pp. 293–294.

[100] M. Karr, *Summation in finite terms*, Journal of the ACM, 28 (1981), pp. 305–350.

[101] M. Kauers and C. Schneider, *Symbolic summation with radical expressions*, in ISSAC '07: Proceedings of the 2007 international symposium on Symbolic and algebraic computation, New York, NY, USA, 2007, ACM, pp. 219–226.

[102] P. H. Knowles, *Integration of liouvillian functions with special functions*, in Proceedings of the fifth ACM symposium on Symbolic and algebraic computation, Symposium on Symbolic and Algebraic Manipulation, New York, 1986, ACM, pp. 179 – 184.

[103] D. E. Knuth, *The analysis of algorithms*, in In proceedings of the International Congress of Mathematicians, vol. 3, 1970, pp. 269–274.

[104] ———, *The art of computer programming, volume 2: seminumerical algorithms*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 ed., 1997.

[105] E. Kolchin, *Differential Algebra and Algebraic Groups*, Academic Press, New York, 1973.

[106] J. J. Kovacic, *An algorithm for solving second order linear homogeneous differential equations*, Journal of Symbolic Computation, 2 (1986), pp. 3 – 43.

[107] ———, *An algorithm for solving second order linear homogeneous differential equations*, CCNY Colloquium lecture, (2001).

[108] H. T. Kung, *On computing reciprocals of power series*, Numer. Math., 22 (1974), pp. 341–348.

[109] Laderman, *A noncommutative algorithm for multiplying* 3×3 *matrices using 23 multiplications*, Bull. AMS, 82 (1976), pp. 126–128.

[110] J. Lagaria, V. Miller, and A. Odlyzko, *Computing* $\pi(x)$*: The meissel-lehmer method*, Mathematics of Computation, 44 (1985), pp. 537–560.

[111] J. Lagarias and A. Odlyzko, *Computing* $\pi(x)$*: An analytic method*, Journal of Algorithms, 8 (1987), pp. 173–191.

[112] D. H. Lehmer, *An extended theory of lucas' functions*, The Annals of Mathematics, 31 (1930), pp. 419–448.

[113] ——, *An inversive algorithm*, Bulletin of the American Mathematical Society, 38 (1932), pp. 693–694.

[114] ——, *Euclid's algorithm for large numbers*, The American Mathematical Monthly, 45 (1938), pp. 227–233.

[115] ——, *On arccotangent relations for* π, Amer. Math. Monthly, 45 (1938), pp. 657–664.

[116] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, *Factoring polynomials with rational coefficients*, Mathematicsche Annaln, (1982), pp. 515–534.

[117] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard, *The factorization of the ninth fermat number*, Mathematics of Computation, 61 (1993), pp. 319–349.

[118] H. W. Lenstra, Jr., *Factoring integers with elliptic curves*, The Annals of Mathematics, 126 (1987), pp. 649–673.

[119] M. Mendès France and G. Tenenbaum著，姚家燕译, 素数论, 研究生数学丛书, 清华大学出版社, 北京, 10 2007.

[120] M. Martin, *Re: Baillie-psw - which variant is correct?*, 2004. http://groups.google.com/group/sci.crypt/msg/48ec324b9fc9f866.

[121] M. McClellan, *The exact solution of systems of linear equations with polynomial coefficients*, Journal of the Association for Computing Machinery, 20 (1973), pp. 563–588.

[122] W. R. Mekwi, *Iterative methods for roots of polynomials*, master's thesis, Exeter College, University of Oxford, 2001.

[123] R. T. Moeck and J. H. Carter, *Approximate algorithms to derive exact solutions to systems of linear equations*, in Proc. EUROSAM '79, vol. 72 of Lecture notes in computer science, Springer-Verlag, 1979, pp. 65–73.

[124] R. T. Moenck, *Fast computation of gcd's*, in In Proceedings of the Fifth Annual ACM Symposium on the Theory of Computing, Austin TX, ACM Press, 1973, pp. 142–151.

[125] P. L. Montgomery, *Modular multiplication without trial division*, Mathematics of Computation, 44 (1985), pp. 519–521.

[126] ——, *Speeding the pollard and elliptic curve methods of factorization*, Mathematics of Computation, 48 (1987), pp. 243–264.

[127] M. A. Morrison and J. Brillhart, *A method of factoring and the factorization of* f_7, Mathematics of Computation, 29 (1975), pp. 183–205.

[128] J. Moses, *Symbolic Integration*, PhD thesis, Massachusetts Institute of Technology, 1967.

[129] ——, *Symbolic integration: the stormy decade*, in Proceedings of the second ACM symposium on Symbolic and algebraic manipulation, Symposium on Symbolic and Algebraic Manipulation, New York, 1971, ACM, pp. 427 – 440.

[130] J. Moses and D. Y. Y. Yun, *The EZ GCD algorithm*, Proc. ACM73, (1973), pp. 159–166.

[131] M. Newman, *Integral matrices*, Academic Press, New York and London, 1972.

[132] J. Nolan, *Analytical differentiation on a digital computer*, master's thesis, Massachusetts Institute of Technology, 5 1953.

[133] S. Pagliarulo, *Stu's pi page*. http://home.istar.ca/~lyster/otherconstants.html.

[134] V. Pan, *How can we speed up matrix multiplication?*, SIAM Review, 26 (1984).

[135] V. Y. Pan, *Methods of computing values of polynomials*, Russian Mathematical Surveys, 21 (1966), pp. 105–136.

[136] D. S. Parker, *A randomizing butterfly transformation useful in block matrix computations*, tech. report, Computer Science Department, University of California, 1995.

[137] P. Paule, *Greatest factorial factorization and symbolic summation*, Journal of Symbolic Computation, 20 (1995), pp. 235–268.

[138] M. Petkovsek, H. Wilf, and D. Zeilberger, $A{=}B$, A K Peters, Ltd., 1996.

[139] R. Pirastu, *On Combinatorial Identities: Symbolic Summation and Umbral Calculus*, PhD thesis, RISC, J. Kepler University Linz, 1996.

[140] J. M. Pollard, *Theorems on factorization and primality testing*, Mathematical Proceedings of the Cambridge Philosophical Society, 76 (1974), pp. 521–528.

[141] ———, *A monte carlo method for factorization*, BIT Numerical Mathematics, 15 (1975), pp. 331–334.

[142] C. Pomerance, *Analysis and comparison of some integer factoring algorithms*, in Computational Methods in Number Theory, Mathematisch Centrum, 1982, pp. 89–139.

[143] ———, *Are there counterexamples to the baillie-psw primality test?*, 1984. http://www.pseudoprime.com/dopo.pdf.

[144] C. Pomerance, J. L. Selfridge, and S. S. Wagstaff, Jr., *The pseudoprimes to* $25{\cdot}10^9$, Mathematics of Computation, 35 (1980), pp. 1003–1026.

[145] M. O. Rabin, *Probabilistic algorithm for testing primality*, Journal of Number Theory, 12 (1980), pp. 128–138.

[146] H. Riesel, *Prime Numbers and Computer Methods for Factorization*, PM 57, Boston; Basel; Stuttgart: Birkhäuser, 1985.

[147] R. H. Risch, *The problem of integration in finite terms*, Transactions of the American Mathematical Society, 139 (1969), pp. 167–189.

[148] ———, *The solution of the problem of integration in finite terms*, Bulletin of the American Mathematical Society, 76 (1970), pp. 605–608.

[149] ———, *Algebraic properties of the elementary functions of analysis*, Robert H. Risch, 101 (1979), pp. 743–759.

[150] A. Rockett and P. Szusz, *Continued fractions*, World Scientific, 1992.

[151] M. Rothstein, *A new algorithm for the integration of exponential and logarithmic functions*, Proceedings of the 1977 MACSYMA Users Conference, (1977), pp. 263–274.

[152] B. D. Saunders, *An implementation of kovacic's algorithm for solving second order linear homogeneous differential equations*, Proceedings of the fourth ACM symposium on Symbolic and algebraic computation, (1981), pp. 105–108.

[153] C. Schneider, *Symbolic summation with single-nested sum extensions*, in ISSAC '04: Proceedings of the 2004 international symposium on Symbolic and algebraic computation, New York, NY, USA, 2004, ACM, pp. 282–289.

[154] A. Schoenhage and V. Strassen, *Schnelle multiplikation groeer zahlen*, Computing, (1971), pp. 281–292.

[155] A. Schönhage, *Schnelle berechnung von kettenbruchentwicklungen*, Acta Informatica, 1 (1971), pp. 139–144.

[156] J. T. Schwartz, *Fast probabilistic algorithms for verification of polynomial indentities*, Journal of the ACM, 27 (1980), pp. 701–717.

[157] S. M. Sedjelmaci, *Jebelean–weber's algorithm without spurious factors*, Information Processing Letters, 102 (2007), pp. 247–252.

[158] W. M. Seiler, *Computer algebra and differential equations - an overview.* http://iaks-www.ira.uka.de/iaks-calmet/werner/Papers/CADE.ps.gz.

[159] J. L. Selfridge and A. Hurwitz, *Fermat numbers and mersenne numbers*, Mathematics of Computation, 18 (1964), pp. 146–148.

[160] M. F. Singer, *Liouvillian solutions of n-th order homogeneous linear differential equations*, American Journal of Mathematics, 103 (1981), pp. 661–682.

[161] ——, *Liouvillian solutions of linear differential equations with liouvillian coefficients*, Journal of Symbolic Computation, 11 (1991), pp. 251 – 273.

[162] J. R. Slagle, *A heuristic program that solves symbolic integration problems in freshman calculus*, Journal of the ACM, 10 (1963), pp. 507 – 520.

[163] R. Solovay and V. Strassen, *A fast monte-carlo test for primality*, SIAM Journal on Computing, 6 (1977), pp. 84–85.

[164] J. Sorenson, *Two fast gcd algorithms*, Journal of Algorithms, 16 (1994), pp. 110–144.

[165] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.

[166] ——, *The computational complexity of continued fractions*, SIAM journal on Computing, 12 (1983), pp. 1–27.

[167] The GMP team, *The gnu mp bignum library.* http://gmplib.org/.

[168] ——, *The GNU Multiple Precision Arithmetic Library*, 4.2.3 ed., July 2008.

[169] L. N. Trefethen and D. Bau著，陆金甫，关治译, 数值线性代数, 人民邮电出版社, 2006.

[170] M. van der Put and M. F. Singer, *Galois theory of linear differential equations*, 科学出版社, 北京, 2007.

[171] G. Villard, *Some recent progress in exact linear algebra and related questions*, in ISSAC'07, Waterloo, Ontario, Canada, July 29-August 1 2007.

[172] J. von zur Gathen, *Hensel and newton methods in valuation rings*, Mathematics of Computation, 42 (1984), pp. 637–661.

[173] ——, *Functional decomposition of polynomials: the tame case*, Journal of Symbolic Computation, 9 (1990), pp. 281–299.

[174] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, Cambridge University Press, 2nd ed., 2002.

[175] J. von zur Gathen and V. Shoup, *Computing frobenius maps and factoring polynomials*, Computational Complexity, 2 (1992), pp. 187–224.

[176] Z. Wan, *An algorithm to solve linear systems exactly using numerical methods*. preprint submitted to elsevier science, 12 2005.

[177] P. S. Wang, *An improved multivariate polynomial factoring algorithm*, Math. Comp., 32 (1978), pp. 1215–1231.

[178] P. S. Wang and L. P. Rothschild, *Factoring multivariate polynomials over the integers*, Math. Comp., 29 (1975), pp. 935–950.

[179] X. Wang and V. Pan, *Acceleration of euclidean algorithms and extensions*, in ISSAC'02, 2002.

[180] K. Weber, *The accelerated integer gcd algorithm*, ACM Transactions on Mathematical Software, 21 (1995), pp. 111–122.

[181] E. W. Weisstein, *From mathworld–a wolfram web resource: Horner's rule*. http://mathworld.wolfram.com/HornersRule.html.

[182] ———, *From mathworld–a wolfram web resource: Machin-like formulas*. http://mathworld.wolfram.com/Machin-LikeFormulas.html.

[183] ———, *From mathworld–a wolfram web resource: Pi formulas*. http://mathworld.wolfram.com/PiFormulas.html.

[184] D. Wiedemann, *Solving sparse linear equations over finite fields*, IEEE transactions on infomation theory, 32 (1986), pp. 54–62.

[185] H. C. Williams, *A $p+1$ method of factoring*, Mathematics of Computation, 39 (1982), pp. 225–234.

[186] S. Winograd, *A new algorithm for inner product*, IEEE Trans. Comp., 17 (1968), pp. 693–694.

[187] C.-K. Yap, *Fundamental problems of algorithmic algebra*, Oxford University Press, 2000.

[188] D. Y. Y. Yun, *On square-free decomposition algorithms*, in In Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation ISSAC'76, R. D. Jenks, ed., Yorktown Heights NY, ACM Press, pp. 26–35.

[189] H. Zassenhaus, *On hensel factorization*, Journal of Number Theory, 1 (1969), pp. 291–311.

[190] R. Zippel, *Probabilistic algorithms for sparse polynomials*, in Lec. Notes Comp. Sci., London, UK, 1979, Springer-Verlag, pp. 216–226.

[191] ———, *Effecive Polynmial Computation*, Kluwer Academic Publishers, Boston/Dordrecht/London, 1993.

索引

致谢

两年过去, maTHμ 从无到有, 一路走来. 项目团队最为欣慰的事莫过于看着我们的“幻想”正一点点变为现实. 作者感谢 maTHμ 项目指导老师饶辉和法国教授 Jacques Peyrière 对项目的精心指导. 感谢清华大学数学科学系李建国, 白峰杉, 李津, 扈志明, 文志英, 卢旭光等老师, 校团委阳波老师, 校科创中心及校科协对项目提供的支持. 感谢中科院数学机械化实验室李子明老师的有益讨论.

本文第 1, 5, 14 章由张翔撰写, 第 2 – 4, 15 – 16 章由李超撰写, 第 6 章由张龙撰写, 第 7 – 13 章由阮威撰写. 李超负责统筹了全文并撰写了附录.