数理解析研究所講究録
986巻 1997年 34-40

Replicable function の Risa/Asir による計算

(株) 富士通研究所 HPC 研究センター 野呂正行 (Noro Masayuki)

1. Monster 単純群, Modular 函数

ここでは, 計算の対象となる replicable function について Alexander et al. に従って概説する.

位数最大の散在型単純群 Monster の巡回部分群の共役類 $< m >$ に対し, Conway-Norton [1] により Modular 函数 $j_{<m>}(z)$ が empirical に対応づけられた. その対応は, $j_{<m>}(z)$ の Fourier 係数がある表現 (head representation) の trace で書けるというものである. 特に, 単位元には elliptic modular 函数 $J(z) = j(z) - 744$ が対応する.

$J(z)$ は, Hecke operator T_n の作用に対して次の函数等式を満たす.

$$J|T_n = \frac{1}{n} \sum_{ad=n, 0 \le b < d} J((az+b)/d) = P_n(J(z))/n$$

この拡張として, 函数 f およびその "replicate function" $f^{(a)}$ を含む式

$$P_n(f(z))/n = \frac{1}{n} \sum_{ad=n, 0 \le b < d} f^{(a)}((az+b)/d)$$

を考え, これを満たす $f^{(a)}$ が存在するとき f を replicable function と定義する. ここで $P_n(t) \in Z[a_1, \cdots, a_{n-1}][t]$ は, $P_n(f(z)) = q^{-n} \bmod qZ[q]$ を満たす unique な多項式である. この式の右辺を $f|\hat{T}_n$ と書き, twisted Hecke operator と呼ぶ.

2. Replicable function

前節の定義のより構成的な言い換えが Norton [4] により与えられている.

$$f(z) = q^{-1} + a_0 + a_1 q + a_2 q^2 + \cdots, q = e^{2\pi i z}$$

とし,

$$F(y, z) = log(f(y) - f(z))$$

の展開を考えると a_i の多項式 $H_{m,n}$ が存在して

$$F(y, z) = log(r^{-1} - q^{-1}) - \sum_{m,n=1}^{\infty} H_{m,n} q^m r^n,$$

$$r = e^{2\pi iy}, H_{n,1} = a_n$$

となることが言える. この時,

f is replicable $\Leftrightarrow H_{m,n} = H_{r,s}$ whenever $nm = rs$ and $gcd(m,n) = gcd(r,s)$

であることが, Norton [4] により示された. $H_{m,n}$ は

$$(r+s)H_{r,s} = \sum_{m=1}^{r-1}\sum_{n=1}^{s-1}(r+s-m-n)a_{r+s-m-n-1}H_{m,n}$$

なる recurrence relation を持つ. これと, 上の同値より, 無限個の変数 a_i に関する無限個の代数方程式が得られる.

3. McKay's problem

前節で得られた方程式系は, 一見変数が無限個で手に負えないように見えるが, f が replicable ならば, $H_{m,n}$ は

$a_i, i \in B = \{1,2,3,4,5,7,8,9,11,17,19,23\}$ (Norton Basis[4])

の多項式として書けることが Norton [4] により証明されている. よって解くべき方程式は a_i $(i \in B)$ に対する無限個の方程式と考えてよい. ここでは, この方程式の解を求める問題を McKay's problem と呼ぶことにする.

4. 実験 (試行錯誤)

我々は, McKay's problem に対し次のような実験を行った. 方程式の生成は, はじめは McKay による Maple program により, 後に Asir 上で書いた program により行った.

4.1. 特殊な場合の解を求めてみる

Maple プログラムで生成した 7 本の方程式をもとに, 5 (=12-7) つの変数を 0 と置いた方程式のグレブナ基底の計算を行ってみた. 具体的には, McKay の示唆により $a_1 = a_2 = a_3 = a_4 = a_8 = 0$ とおいて計算してみたが, 結果を得ることはできなかった.

4.2. Replication order がある場合

f に対し, f の k-th replication power を次で定義する.

$$f(z) \mapsto f^{(k)}(z) = q^{-1} + \sum_{i=1}^{\infty} a_i^{(k)} q^i$$

$$a_i^{(k)} = k\sum_{d|k}\mu(d)H_{\frac{k}{d},dki}$$

$f^{(gcd(k,n))} = f^{(k)} (\forall k \geq 1)$ なる n が存在する時, n を replication order と呼ぶ.

これまで発見された replicable function は全て replication order を持つことから replication order がある場合を考えてみた. 最も簡単な場合として, n が奇数ならば, $f^{(2)} = f$ となる.

これは odd replication order の場合と呼ばれる. この場合 a_1, a_2, a_3, a_5 でそれ以外の変数が書ける. 即ち, これらの変数以外の各変数 a_i に対し, a_i について 1 次で a_i の係数が有理数である方程式が存在する.
この場合に関しては全ての整数解が modulo 3 の解から出発して, Hensel 構成により得られている (Norton [4]).
我々は odd replication order の場合に対し, 全ての複素数解を求めることに挑戦してみた. 変数は $\{a_1, a_2, a_3, a_5\}$ の 4 つであるが, 実際に何個の方程式があれば十分であるかは不明である. ここでは, 20 個の方程式を生成して実験してみた. この場合の方程式を E と書くことにする. E の規模は次の通りである.

- 方程式の数
  20
- E の各元に対する項の数
  [64,75,75,78,96,105,108,123,179,179,201,208, 214,252,269,282,323,329,815,1183]
- E の各元の全次数
  [7,7,7,7,8,8,8,9,9,9,9,9,10,10,10,11,11,11,15,17]

5. DRL グレブナ基底 G の計算

結果的には, Sparc Ultra 1 上で, 約 25 日かけて計算に成功した. その結果から, G は 0 次元 システムでないことが判明した. このことは方程式が不足であるか, または元々 0 次元でないかのいずれかであることを示している. modular グレブナ基底計算により, $\{a_1 = 0\}$ なる 高次元成分の存在が推定された. そこで $\{a_1 = 0\}$ なる 成分の除去を行ってみた. これは $G \cup \{ta_1 - 1\}$ の, $t > \{a_1, a_2, a_3, a_5\}$ なる消去順序によるグレブナ基底計算を行うことにより可能である.

```
[100] Elim = gr(cons(t*a_1-1,G),[t,a5,a3,a2,a1],[[0,1],[0,4]])$
```

この計算は PentiumPro 200MHz マシン上で 1 分足らずで終了する. Elim から t を含まない元を集めると $\{a_1 = 0\}$ を除いた零点集合に対応するシステム G_0 が得られる. すると G_0 が 0 次元システムであることが分かった.
G_0 の零点は radical の素イデアル分解で容易に求まる. (同じマシン上で 130 秒)

```
[101] P = primedec(G0,[a5,a3,a2,a1])$
```

6. 素イデアル分解の結果 — 解の表示

以下は, 素イデアル分解の結果である.

```
[a1-27,a2-86,a3-243,a5-1370], [a1-12,a2-28,a3-66,a5-258],
[a1+1,a2-2,a3-1,a5+2], [a1+1,a2+1,a3-1,a5+1], [a1+1,a2-1,a3+1,a5-2],
[a1+1,a2,a3,a5], [a1+1,a2-4,a3+3,a5-11], [a1+1,a2-2,a3+2,a5-5],
[a1+1,a2-1,a3,a5-2], [a1-9,a2+4,a3,a5-2], [a1-9,a2-10,a3+30,a5+25],
[a1-9,a2-19,a3-42,a5-146], [a1+6,a2-20,a3-15,a5],
```

```
[a1-196884,a2-21493760,a3-864299970,a5-333202640600],
[a1-8,a2-22,a3-42,a5-155], [a1-1,a2,a3,a5], [a1-1,a2-2,a3-2,a5-5],
[a1-1,a2-1,a3-1,a5-2], [a1-1,a2-1,a3-1,a5-3], [a1-1,a2-1,a3,a5-1],
[a1-1,a2-1,a3-2,a5-3], [a1-1,a2,a3-1,a5-1], [a1-1,a2+1,a3-1,a5],
[a1-1,a2-2,a3-3,a5-5], [a1-1,a2-4,a3-6,a5-10],
[a1-51,a2-204,a3-681,a5-5135], [a1-3,a2-3,a3-6,a5-13],
[a1-3,a2-4,a3-7,a5-17], [a1-3,a2+1,a3,a5+1], [a1-3,a2-1,a3-3,a5-6],
[a1-3,a2-2,a3+3,a5], [a1-3,a2-5,a3-9,a5-20], [a1-5,a2+2,a3,a5+1],
[a1-5,a2-8,a3-16,a5-44], [a1-2,a2-1,a3-3,a5-5], [a1-2,a2-1,a3-1,a5-3],
[a1-2,a2-8,a3+5,a5+10], [a1-2,a2-4,a3-5,a5-14], [a1-2,a2-2,a3-3,a5-7],
[a1-2,a2-2,a3-4,a5-7], [a1-2,a2-1,a3-2,a5-4], [a1-2,a2-3,a3-5,a5-10],
[a1-783,a2-8672,a3-65367,a5-1741655], [a1-7,a2-14,a3-29,a5-92],
[a1-4,a2-7,a3-13,a5-33], [a1-4,a2-5,a3-10,a5-25],
[a1-54,a2+76,a3+243,a5+1384], [a1-6,a2-10,a3-21,a5-61],
[a1-6,a2-6,a3-15,a5-41], [a1-17,a2-46,a3-116,a5-533],
[a1+3,a2-2,a3,a5-5], [a1-134,a2-760,a3-3345,a5-39350],
[21*a1^4-154*a1^3+52*a1^2-150*a1+391,2*a2+a1-1, a3,8*a5-a1^2+8*a1-7]
```

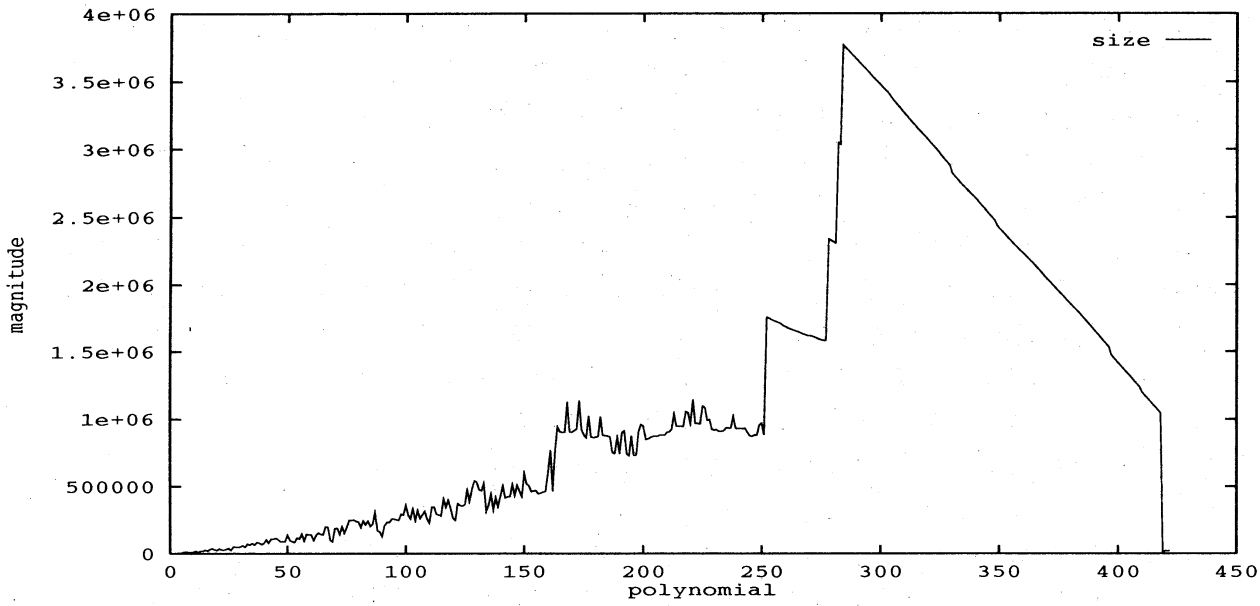
```
[a1-196884,a2-21493760,a3-864299970,a5-333202640600]
```

が elliptic modular 函数に対応する解である.
最後の解のみ非整数 (非有理数解) であるが, この場合に対し, 方程式を更に生成して付加することで, 解でないことが分かった.
最後に, $\{a_1 = 0\}$ に対応する解は, 同様に方程式を更に生成して付け加えることで 0 次元化され求めることができた. 求めた解は全て整数解であり, Norton [4] による整数解以外に複素数解がないことが示せた.

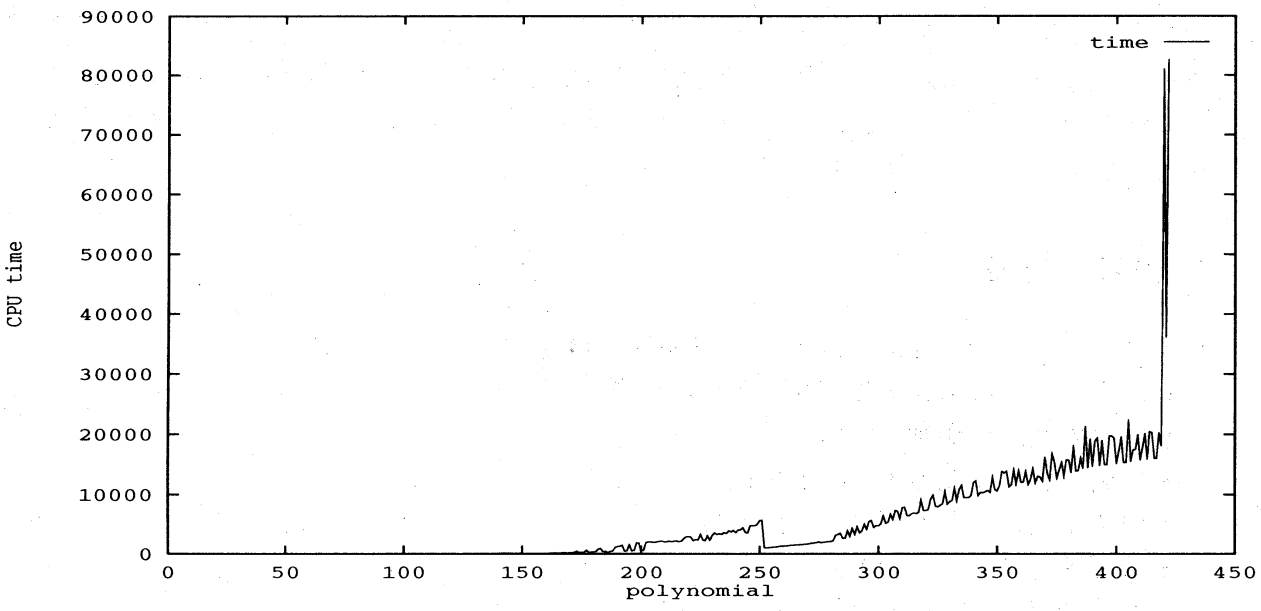
7. 計算の詳細

- 使用マシン $\cdots$ Sparc Ultra-1 (170MHz, 448MB)
- 設定 $\cdots$ $Multiple = 2$, $Demand = 1$, 斉次化あり
- 結果の基底の個数 $\cdots$ 51
- 生成された中間基底 $\cdots$ 443 個
- 中間基底の合計サイズ $\cdots$ 91MB
- 所要メモリ量 $\cdots$ 440MB
- 所要時間 $\cdots$ 2×10^6 秒 (CPU) + 6×10^4 秒 (GC)
- 係数の bit 長の最大値 $\cdots$ 18880 (中間基底), 289 (結果)
- 係数の bit 長の和の最大値 $\cdots$ 3771406 (中間基底), 22378 (結果)
- normal form 計算の最長時間 $\cdots$ 8×10^4 秒
- 斉次化なしでの計算 $\cdots$ メモリ不足でストップ
- Multiple なし, Demand なし $\cdots$ メモリ不足に陥る

8. 基底のサイズ, 計算時間



各中間基底の係数の bit 長の和



各中間基底の計算時間 (秒)

9. 中国剰余定理

実験中, 結果が小さいことを期待して, 次の手順 (佐々木, 竹島 [5]) によりグレブナ基底候補を計算してみた.

1. 多くの素数 ($\simeq$ 1 マシンワード) に対し modular グレブナ基底を計算
2. 中国剰余定理で結合
3. 係数を有理数化
4. 未使用の素数に対し グレブナ基底かどうか調べる
5. 実際に有理数上で グレブナ基底かどうか調べる
6. 得られた グレブナ基底が, 入力多項式を 0 に正規化するかどうか調べる

これにより, 元のイデアルを含むイデアルを生成する グレブナ基底は求まる.
$\Rightarrow$ 30 個弱の素数で十分 (数時間で計算終了)
しかし, ここで得られたイデアルの零点は, 一般には元のイデアルの零点の部分集合になることしか言えない. これは, 全ての解を求めるという今回の実験においては不十分である. 解が一致することを示すためには, イデアル間の逆の包含関係を示すことが十分であるが, これは, 多項式 $f_1, \cdots, f_n, g$ に対し, $g = \sum h_i f_i$ なる多項式 h_i を求めることに帰着される. このためには有理数上の グレブナ基底計算時に, ヒストリを残せば可能だが, これは余分なコストを生じさせる.
効率化のため, modular 計算 + 中国剰余定理, あるいは modular 計算 + 未定係数法 (線形代数) に挑戦してみたが, 実際に計算してみると, 項の数が莫大 ($\simeq 10^4$) となりなんらかの工夫が必要であることが分かった.

10. 反省および今後の計画

今回の実験により現在の版の グレブナ基底計算能力の限界が見えた. グレブナ基底計算をより実用的なものにするためにはさらなる効率化が必要と考えられる. 0 次元システムに関しては, change of ordering に modular 計算導入することによりかなりの効率化が達成できたが, 0 次元でも, change of ordering に持ち込む前処理としての DRL グレブナ基底計算が困難な場合がある. 今後は, Buchberger Algorithm における normal form 計算などの, 並列分散計算も採り入れた効率化を図る.

また, 得られた結果が本当に入力多項式から生成されているかどうかを保証するためには, 生成関係式を与える必要がある. これを計算することは, グレブナ基底を直接計算するより遥かに困難である. この生成関係式を, やはり modular 計算などを援用することにより効率的に計算することも目標とする.

replicable function の計算に関しては, 次のようなことが考えられる.

- odd level 以外の場合の計算
  $f^{(N)} = f$ で, N が大きい場合に, 整数解以外の解が見つかるかどうかが興味ある問題となっている.
- completely replicable function の計算
  replicable function に, さらに強い条件をつけた, completely replicable function も, やはり代数方程式で記述される対象で, これを解くことも要請されている.

参 考 文 献

[1] Conway, J.H., Norton, S., Monstrous moonshine, Bull. London Math. Soc. 11, 308-339(1979).

[2] Ford, D.J., Norton, S., McKay, J., More on replicable functions, Comm. Alg. 22, 5175-5193(1994).

[3] Alexander, D., Cummins, C., McKay, J., Simons, C., Completely replicable functions, Groups, Combinatorics and Geometry (Liebeck, Saxl, eds.), LMS monograph series, 165, Cambridge University Press, 87-98(1992).

[4] Norton, S., More on moonshine, Computational group theory, (Atkinson, ed.) 185-193(1984).

[5] Sasaki, T., Takeshima, T., A Modular Method for Gröbner-basis Construction over Q and Solving System of Algebraic Equations, J. of Information Processing, Vol. 12, No. 4, 371-379(1989).

A. メモリに制限がある場合の グレブナ基底計算

`Asir` においては, グレブナ基底計算を制御するオプションは `dp_gr_flags()` で設定する. それらのオプションの中で, 今回の実験で特に有効であったものについて説明する.

- `Multiple`=N
  normal form 計算において, 頭係数の bit 長が (magnitude) が N 倍になったら整数 content を取ることを指示する. 通常は 0 が設定されていて, その場合には, normal form になった時点で整数 content をとるが, normal form までの reduction の回数が多い場合などにおいては, 結果の係数が小さい場合でも途中の係数膨張によりメモリ不足に落ちいる場合がある. この設定は cyclic, McKay など問題が代数的にきれいな性質を持っている場合などに特に有効で, 今回は $N = 2$ で計算した.
- `Demand`=*directoryname*
  整数係数の中間基底を指定ディレクトリに保存し, メモリ上に残さない. reduction の都度, ディスクから必要な中間基底のみを読み出す.
  - 小さい問題では overhead が大きいが, McKay などの場合極めて有効.
  - 計算途中でのマシントラブルより計算が中断しても, 再開が容易 (これは次期バージョンからサポートされる).