4/16

数理科学 II (2012年度)

概要

◎ 計算代数 (数式処理)

↕ computer algebra (computational)

↘ formula manipulation

有限体 ($\mathbb{Z}/p\mathbb{Z}$), $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ 上の
($\pm n$) 多項式の諸算法と応用.

◎ 担当と扱うテーマ

前半 (照井)
- 多項式環, アルゴリズム, 多項式の四則演算
- 計算量
- 1変数 / 多変数多項式の因数分解

後半 (田島) グレブナー
- 多項式環上の Gröbner 基底
- 微分作用素環と Gröbner 基底.

◎ 前提とする知識 (必要 → ○, 十分 → △)

○ 微積分, 線形代数.
○ 群 / 環 / 体の基本的事項
△ 計算機の知識, 経験: あれば役立つが, 現時点ではなくても困らない (必要事項は授業時に説明)
但し, 授業内容に応じて概念があれば積極的に触れてほしい.

◎ 参考書 (詳しくはシラバスを参照).

- von zur Gathen and Gerhard:
  - 現代的な教科書.
  - 計算代数を専門にする人は持っておいた方がよい.
  - 計算量解析がかなり詳しい.
  - 現在, 入手が難しいみたい

- Geddes, Czapor & Labahn
  - Mapleの開発者らによる本（Maple本）
  - 扱っている内容が割と幅広い　（Amazon.co.jpで 1円くらい）
  - 値段が、高かったが、今はそうでもないようだ

◎ 数式処理システム（Computer Algebra System, CAS）

- Mathematica（Wolfram Research）
  - 今、世界で最も広く使われているものの一つ
  - 大学でも使えます
  - 自宅でも使えます（Mathematica for Students）
  - 「計算機演習」（数学科2年次）TA募集中！
- Maple（Maplesoft）
  - Mathematicaと同じく、よく使われている。
  - 日本の販売会社「サイバネットシステム」がMaplesoftを買収。

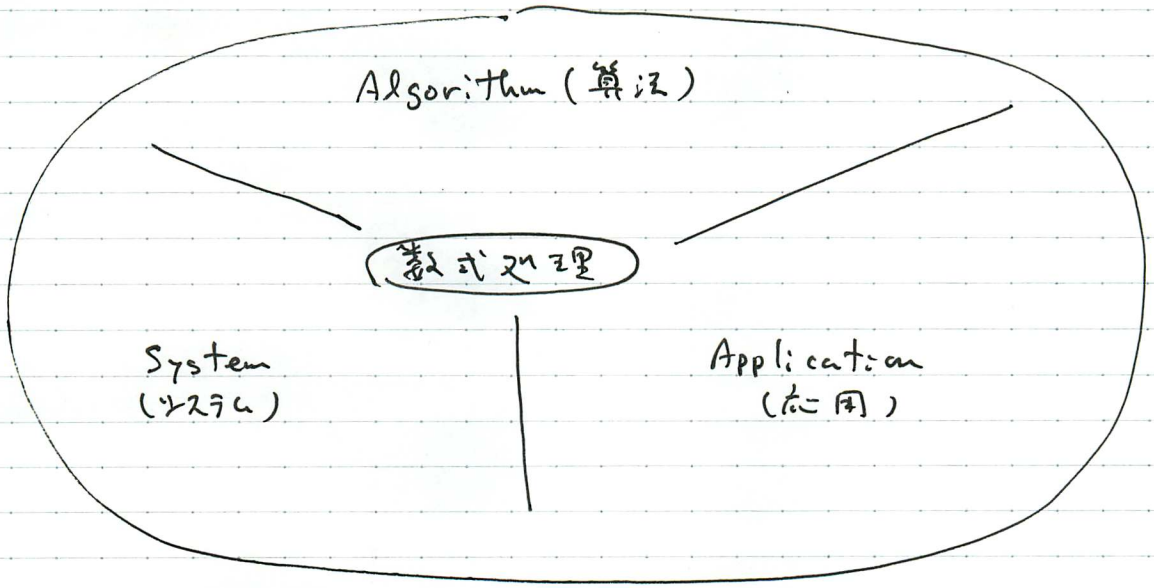
フリーのソフトウェアもいろいろある。最も代表的なもの：

- REDUCE
  - Mathematicaより前に最も普及したシステムの一つ（口タ大）
  - 当時は商品、今はフリーソフト
  - ライブラリが豊富
- Maxima
  - MITで開発されたMACSYMAの流れをくむ。
  - 1960－70年代の最も先進的な算法を採り入れたシステム。
  - 今はフリーソフト。
- Risa/Asir
  - 1990年代初めから富士通研究所で開発。現在は神戸大が中心。
  - Gröbner基底計算が強力。
- Sage
  - 最近開発が進んでいる統合型数式処理システム。

⇒ KNOPPIX/Mathを使って触れてみよう！

  - 既存のいろいろなシステムをプログラミング言語Pythonを用いてつなぎ合わせる。
  - Webブラウザから利用可。オンラインで使えるサービスもあり。

◎ 数式処理（研究）の3手柱（by 伝え手）



- Algorithm ： 数学力
- System ： 計算機の技術、ノウハウ
- Application ： 応用分野の知見

数式まわり、数学者、計算機科学者、応用分野の科学者/技術者のﾁｰﾑ（collaboration）が集んでいる。

skip
◎ 数値計算との比較。

比較項目	数式処理	数値計算
係数演算の精度	厳密	近似
計算手法の厳密さ	◎	○ or △（誤差解析、検証が必要）
計算の効率	○	◎
ユーザ/開発者、研究者の規模	△	◎

NO. 4

DATE 16 APR 2012

◎ 数値計算との比較（例）

例題	数式処理	数値計算
π	π	3.1415926… ◎
		↑ 有限桁で近似
		↓
$1/7$	$1/7$	0.142857… ◎
$1\div3\times3$	1	0.9999… ◎

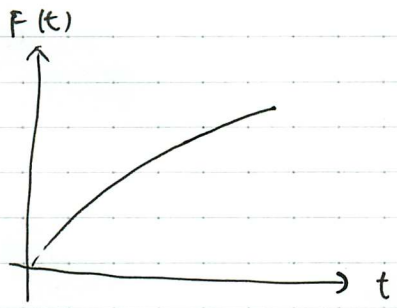
浮動小数点数（浮動小数）

（実数を有限桁で近似しそれの加減乗除に関して次ぎつぎも閉じていない。計算のたびに誤差が生ずる。）

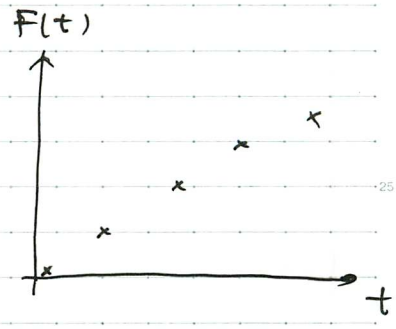
プログラム

微分方程式の解

$$\begin{cases} \dfrac{dF(t)}{dt} = f(t). \\ F(0) = a \end{cases}$$



（解けれが）解析的に解を求める



時刻 t を離散的にとって解を求める。

CHECK

DATE 4.22.2012 NO. 5

4/23

多項式演算 (polynomial arithmetic)

D: 代数系 (数のようなものの集合).
以後, D は 可換環とする (などで)

◎ 記法

$u(x) = u_n x^n + \cdots + u_1 x + u_0 \in D[x]$, $u_n \neq 0$

$n = \deg(u)$: u の次数 (degree)

$u_n = lc(u)$: u の主係数 (the leading coefficient)

($\deg(0) = -\infty$, $lc(0) = 0$ とおく)

$u(x)$ が… モニック (monic) $\overset{\text{def}}{\iff}$ $lc(u) = 1$.

◎ Euclid 整域 (Euclidean domain)

Def (Euclid 整域)

R: 整域が以下をみたすとき, R を Euclid 整域 (Euclidean domain) という:

Euclid 関数 $d: R \longrightarrow \mathbb{N} \cup \{-\infty\}$ が存在し,
$\forall a, b \in R$ に対し, 次で定まる除算 が成りたつ:

もし $b \neq 0$ ならば, $\exists q, r \in R$ s.t.
$a = qb + r$, $d(r) < d(b)$
このとき, q を商, r を剰余という. ◎

※ 上の除算において, q, r は一意とは限らない点に注意!

◎ 一意分解整域 (Unique Factorization Domain : UFD)

Def (一意分解整域)

D: 代数系が… UFD $\overset{\text{def}}{\iff}$

1) D は整域 : $\forall u, v \in D$ $u \neq 0$ かつ $v \neq 0 \Rightarrow uv \neq 0$.
2) $\forall u \in D$ は次のいずれか.
- 単元 (unit) : $\exists u \in D$ s.t. $uv = 1$ (乗法の逆元をもつ)
- 素元 (prime) の積 n 一意的 n 分解される.
$u = p_1 \cdots p_t$, $t \geq 1$, p_i は既約. ◎

以後、D は UFD とする。

Thm (Gauss) D が UFD $\Rightarrow$ $D[x]$ も UFD. □

◎ 原始的な多項式 (primitive polynomials)

Def $u_0, \dots, u_n \in D$ が互いに素.

$\overset{\text{def}}{\iff}$ $u_0, \dots, u_n$ に素元の公約元が存在しない. □

Def (原始的 : primitive)

$u(x) \in D[x]$ s.t. $u(x) = u_n x^n + \dots + u_0 x^0$, $u_n \neq 0$

が原始的 $\overset{\text{def}}{\iff}$ 係数 $u_n, \dots, u_0$ が互いに素. □

* 代数的数の原始多項式も有限個の生成元(代数的数)を係数にもつ原始多項式 (primitive polynomial) と異なるので注意!

Thm (Gauss) D: UFD のとき,
$f, g \in D[x]$ がともに原始的 $\Rightarrow$ fg もまた原始的. □

Lemma D: UFD のとき. $\forall u(x) \in D[x]$ は

$$u(x) = \underset{\in D}{c} \cdot \underbrace{v(x)}_{\text{primitive}} \qquad (1)$$

の形に一意的に表される. □

Def (primitive part, content) 上の lemma の式 (1) において

$$u(x) = pp(u) \quad \text{(primitive part)}$$
$$c = cont(u) \quad \text{(content)}$$
$$\| \\ \gcd(u_n, \dots, u_0)$$

で表す. □

⊘ 多項式の擬除算 (pseudo-division)

(例) 多項式の除算.

$$
\begin{array}{r}
\frac{2}{3}x + \frac{1}{9} \\
3x+1 \overline{)\, 2x^2 + x + 3} \\
2x^2 + \frac{2}{3}x \qquad \\
\hline
\frac{1}{3}x + 3 \\
\frac{1}{3}x + \frac{1}{9} \\
\hline
\frac{26}{9}
\end{array}
$$

D が体でなければ除算が不可能

⟶ D が体でなくとも除算に相当する操作を可能にしたい.

Def (擬除算 : pseudo-division)

$u(x) = u_m x^m + \cdots + u_1 x + u_0$, $\in D[x]$, $u_m \neq 0$

$v(x) = v_n x^n + \cdots + v_1 x + v_0$. $v_n \neq 0$

$m \geq n$

とする. このとき　　　↑ 以後断わらない限り, 同様におく.

u を v で割る擬除算 $\overset{def}{\iff}$ $v_n^{m-n+1} \times u(x)$ を $v(x)$ で割る除算.

擬除算による商, 剰余を, それぞれ擬商 (pseudo-quotient), 擬剰余 (pseudo-remainder) という. ☐

(例) (上の例題) $\begin{cases} u(x) = 2x^2 + x + 3, & m = 2. \\ v(x) = 3x + 1. & n = 1. \end{cases}$

において. $u(x)$ を $v(x)$ で割る擬除算.

$$
\begin{aligned}
v_n^{m-n+1} \times u(x) &= 3^2(2x^2 + x + 3) \\
&= 18x^2 + 9x + 27.
\end{aligned}
$$

```
        6    1
   ┌──────────
3 1 ) 18   9   27
      18   6
      ──────
           3   27
           3   1
           ──────
               26
```

→ $6x+1$: 擬商

↑ 商，剰余とももとの9倍．

↓

→ 26 : 擬剰余．

Skip

例題 $\begin{cases} u(x) = x^3 + 2x^2 - 3x + 4 \\ v(x) = 2x + 1 \end{cases}$ $u(x)$ を $v(x)$ で割る擬除算．

$2^3 \cdot u(x) = 8(x^3 + 2x^2 - 3x + 4)$

$= 8x^3 + 16x^2 - 24x + 32$.

```
        4    6    -15
   ┌──────────────────
2 1 ) 8   16   -2x   32
      8    4
      ──────
           12   -2x
           12    6
           ──────
                -30   32
                -30   -15
                ──────
                      47.
```

擬商 : $4x^2 + 6x - 15$

擬剰余 : 47.

アルゴリズム (算法. algorithm)

ある入力 (式や値) に対し、ある出力 (式や値) を生成するための計算手順 (ステップ).

◎ アルゴリズムの特徴 (要件) (Knuth)

- 停止性 : 有限回のステップで停止すること.
- 正確性 : すべてのステップにおける操作が厳密に定められていること.
- 入力 : 0個以上の入力 (値) をもつこと.
- 出力 : 1個以上の出力 (値) をもつこと.
- 有効性 : 有効な (意味のある) 計算が行われること.

(例) 有効でない例:
整数上の Euclid の互除法に、有理数や無限桁の小数をそれぞれすること.

◎ アルゴリズムの書き方.

Algorithm (アルゴリズムの名前)

入力 : (入力される式や値の定義)
出力 : (出力される式や値の定義)

1.
2.
:
n.

} 各ステップの計算. 上から順番に実行される.

◎ アルゴリズム特有の書式と式

- 変数 : 数や式の値を保持
  (例) p, q, u, w, x, ...
  インデックスがつけられることもある. P_i, P_{i+1}, ...
- 代入 : 変数に式や値を代入
  (例) $u \leftarrow 0$; $u \leftarrow p(x)$;

・論理式：条件の真偽を判定する．

(例) u = 0 ; (uが0に等しいか？)

p and q (pかつqは真か？)

not p (pの否定は真か？ ⇔ pは偽？)

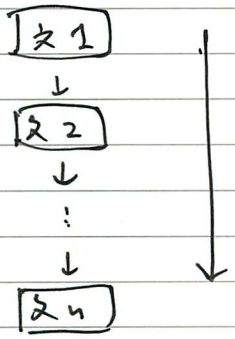
・return：出力を返す式

(例) return p ; (変数pの値を出力として返す)

(11) 制御構造 (control sequence) (岡川．89-10)

① 逐次 (sequence)
② 選択 (selection)
③ 反復 (iteration)

① 逐次：



上から順番に実行．

② 選択：

if文

```
if (論理式 1)
    文1
else if (論理式 2)
    文2
    :
else if (論理式 n-1)
    文n-1
else
    文n
```

上から最初の条件が真になるまで実行.

(論理式 1) が真 → [文 1] を実行.

〃 偽かつ (論理式 2) が真 → [文 2] を実行.

⋮

(論理式 1, …, n−2) が偽かつ (論理式 n−1) が真 → [文 n−1] を実行.

(論理式 1, …, n−1) が偽 → [文 n] を実行.

③ 反復

while 文

```
while (論理式) do
  [文]
```

(論理式) が
- 真 → [文] を (一度) 繰り返して実行.
- 偽 → 次のステップへ移る.

④ 1変数多項式の四則演算のアルゴリズム.

加算

(外い) 数全のゼロ. ⇔ φ

Algorithm (add-unipol-unipol φ)

入力: $\begin{cases} u(x) = u_m x^m + \cdots + u_0 x^0 & \in \mathbb{Q}[x] \\ v(x) = v_n x^n + \cdots + v_0 x^0 & \in \mathbb{Q}[x] \end{cases}$

出力: $w(x) = w_l x^l + \cdots + w_0 x^0 \in \mathbb{Q}[x]$ s.t. $w(x) = u(x) + v(x)$.

$(l \leq \max\{m, n\}$, 但し $w_l \neq 0)$

1. $l \leftarrow \max\{m, n\}$;

2. $i \leftarrow 0$; $w \leftarrow 0$;

3. while $(i \leq l)$ do
   - $a \leftarrow u_i + v_i$;
   - if $(a \neq 0)$
     - $w \leftarrow w + a\, x^i$;
   - $i \leftarrow i + 1$;

4. return w. ☒

減算を「定数倍」と「加算」の組み合わせで定義する.

定数倍

Algorithm (mul-unipol-num ϕ)

入力: $u(x) = u_m x^m + \cdots + u_0 x^0 \in D[x]$.
$c \in D$.
出力: $w(x) = c \cdot u(x) \in D[x]$.

1. $i \leftarrow 0$; $w \leftarrow 0$;
2. while ($i \leq m$) do
   $w \leftarrow w + (c \times u_i) x^i$;
   $i \leftarrow i+1$;
3. return w. ▨

減算

Algorithm (sub-unipol-unipol ϕ)

入力: $\begin{cases} u(x) = u_m x^m + \cdots + u_0 x^0 \in D[x] \\ v(x) = v_n x^n + \cdots + v_0 x^0 \in D[x] \end{cases}$
出力: $w(x) = w_l x^l + \cdots + w_0 x^0 \in D[x]$
s.t. $w(x) = u(x) - v(x)$
($l \leq \max\{m,n\}$, 但し $w_l \neq 0$)

1. return add-unipol-unipol ϕ (
   $u(x)$,
   mul-unipol-num ϕ ($v(x)$, -1)
). ▨

乗算

Algorithm (mul-unipol-unipol ϕ)

入力: $\begin{cases} u(x) = u_m x^m + \cdots + u_0 x^0 \in D[x] \\ v(x) = v_n x^n + \cdots + v_0 x^0 \in D[x] \end{cases}$
出力: $w(x) = w_l x^l + \cdots + w_0 x^0 \in D[x]$
s.t. $w(x) = u(x) \cdot v(x)$ ($l = m+n$).

1. $i \leftarrow 0;\quad w \leftarrow 0;$

2. while $(i \le n)$ do
   $w \leftarrow w + u_i\, x^i \times u(x);$
   $i \leftarrow i + 1;$

3. return w. ▨

擬除算

Algorithm (pdiv-unipol-unipol ϕ) 5/14

入力： $\begin{cases} u(x) = u_m x^m + \cdots + u_0 x^0 \in D[x] \\ v(x) = v_n x^n + \cdots + v_0 x^0 \in D[x] \end{cases}$

出力： $\begin{cases} q(x) = q_l x^l + \cdots + q_0 x^0 \in D[x] & \leftarrow \text{(擬)商}. \\ r(x) = r_k x^k + \cdots + r_0 x^0 \in D[x] & \leftarrow \text{(擬)剰余}. \end{cases}$

s.t. $\begin{cases} v_n^{m-n+1}\, u(x) = q(x)\, v(x) + r(x). \\ k < n. \end{cases}$

1. $r(x) \leftarrow v_n^{m-n+1}\, u(x);\quad q(x) \leftarrow 0;$

2. while $(\deg(r(x)) \geq \deg(v(x)))$ do
   $c \leftarrow lc(r(x)) / lc(v(x));$
   $d \leftarrow \deg(r(x)) - \deg(v(x));$
   $r(x) \leftarrow r(x) - c \cdot x^d \cdot v(x);$
   $q(x) \leftarrow q(x) + c \cdot x^d;$

3. return $(r(x), q(x))$. ▨

計算量べする込！

(付録) 付随する関数. 5/7

Algorithm (deg)

入力： $u(x) = u_m x^m + \cdots + u_0 x^0 \in D[x]$

出力： $c = \deg(u(x)) \in \mathbb{Z}$

1. return m. ▨

Algorithm (lc)

入力 : $u(x) = u_m x^m + \cdots + u_0 x^0 \quad \in D[x]$
出力 : $c = lc(u(x)) \quad \in D$.

1. return u_m.

計算量　Computational complexity

算法の効率を測る上での理論的指標の一つ．

- 時間計算量 (time complexity)：必要な計算のステップ数．
- 空間計算量 (space complexity)：計算に必要な記憶容量．

◎ 計算量の表し方 … 「漸近表示」

$$T: \mathbb{N} = \{0, 1, 2, \dots\} \longrightarrow \mathbb{R}_{\geq 0}$$
$$\cup \qquad\qquad \cup$$
$$n \qquad \mapsto \quad T(n)$$
$$\uparrow$$

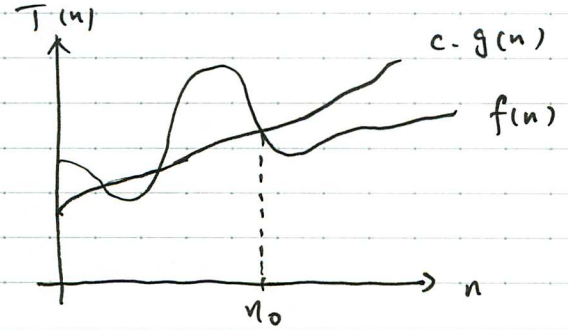
入力データの「サイズ」 … 数の大きさ，数位の桁数．
多項式の次数，…
↳ ここではこれを扱うことが多い．

(例) $f(n)$　n：多項式の次数．
$f(n) = an$，$g(n) = bn^2$，$a > 0$，$b > 0$．
⟶ いずれ g が f を「追い抜く」
⟶ f の方が計算量が「小さい」
他の例：$h(n) = c^n$ $(c >)$，$p(n) = d \log(n)$ $(d > 0)$
等 ◎

◎ 計算量の記法 (notations)

Def (O-notation：漸近的上界)

$$f(n) = O(g(n)) \overset{\text{def}}{\Leftrightarrow} \exists c > 0, \exists n_0 > 0$$
$$\text{s.t.} \quad n \geq n_0 \Rightarrow 0 \leq f(n) \leq c \cdot g(n). \quad \square$$



$f(n) = O(g(n))$

~~(c.f. 数学の O表記：数学では c に依存．ここでは c は挙がる上界．)~~

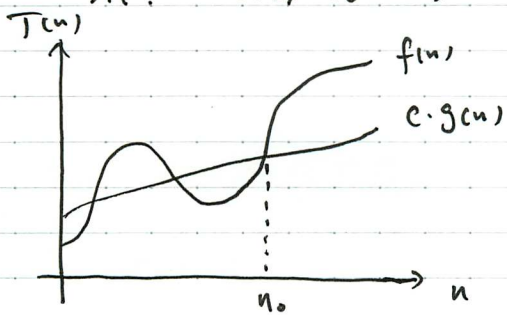
数学でも挙がる上界．
(明解 微分積分．p. 38．注表 2.1)

Def (O-notation)

$$f(n) = O(g(n)) \overset{\text{def}}{\iff} \forall c \geq 0,\ \exists n_0 > 0$$
$$\text{s.t.}\quad n \geq n_0 \Rightarrow 0 \leq f(n) \leq c \cdot g(n).$$
$$\left(\iff \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.\right)$$

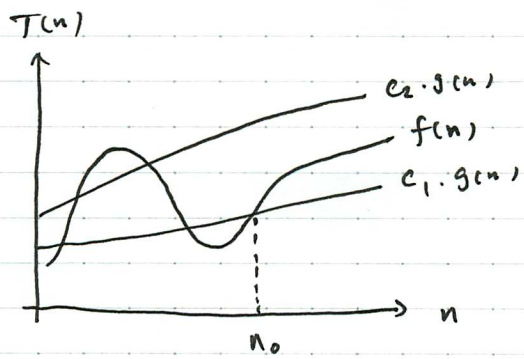
Def (Ω-notation : 漸近的下界)

$$f(n) = \Omega(g(n)) \overset{\text{def}}{\iff} \exists c > 0,\ \exists n_0 > 0$$
$$\text{s.t.}\quad n \geq n_0 \Rightarrow 0 \leq c \cdot g(n) \leq f(n).$$



Def (Θ-notation)

$$f(n) = \Theta(g(n)) \overset{\text{def}}{\iff} \exists c_1 > 0,\ \exists c_2 > 0,\ \exists n_0 > 0$$
$$\text{s.t.}\quad n \geq n_0 \Rightarrow 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$



以後、ここでは基本的に $O(n)$ を用いる.

⑶ 1変数多項式の四則演算（のアルゴリズム）の計算量．

注意
- 係数の加減乗除算を計算ステップの単位とする．
- 理論的な注算の計算量に対し、アルゴリズムの処理は若干冗長な部分が存在し得る．

加算 add-unipol-unipol (u(x), v(x))

1. $l \leftarrow \max\{m, n\}$;　（m, n の定義は p.11 を参照）
2. $i \leftarrow 0$;　$w \leftarrow 0$;
3. while $(i \leq l)$ do　← ループ（繰返し）　$l+1$ 回
   $a \leftarrow u_i + v_i$;　← 加算 1 回（$u_i + v_i \neq 0$ の場合のみ）
   実質 $\min\{m,n\}+1$ 回の加算．
   if $(a \neq 0)$
   $w \leftarrow w + a x^i$;
   $i \leftarrow i+1$;　← ここには係数どうしの加算はないので今回は考慮しない．
4. return w.

⑷ $\min\{m,n\} + 1$回 $= O(\min\{m,n\})$.

定数倍 mul-unipol-num (u(x), c)

1. $i \leftarrow 0$;　$w \leftarrow 0$;
2. while $(i \leq m)$ do　← ループ $m+1$ 回．
   $w \leftarrow w + (c \times u_i) x^i$;
   $i \leftarrow i+1$;　← 乗算 1 回．
3. return w.

⑸ $m+1$回 $= O(m)$.

NO. 18

DATE 5 · 12 · 2012

減算　sub-unipol-unipol (u(x), v(x))

1. return add-unipol-unipol (
　　u(x),
　　mul-unipol-unipol (v(x), -1)
　　).

(加) O(min{m,n})　(乗) O(m^n) = O(m^n).

※加算と同様のアルゴリズムを組める。計算量は O(min{m,n}) に抑えられる。

4/21

乗算　mul-unipol-unipol (u(x), v(x))

1. i ← 0; w ← 0;

2. while (i ≤ n) do ← ループ n+1回
　　w ← add-unipol-unipol (
　　　　w,
　　　　x^i × mul-unipol-num (u(x), v_i)
　　　　);
　　i ← i + 1;

　　(mul-unipol-num: O(m))

　　m+i 次が、項数は m+1.

3. return w.

　　O(min{deg(w), m+i}) だが 定数上は m+1 項の係数を加えているため、O(m).

合計 {O(m) + O(m)} × (n+1) = O(mn)

※高速算法もある。

(擬)除算　pdiv-unipol-unipol (u(x), v(x))

1. r(x) ← mul-unipol-num (v_n^{m-n+1}, u(x));
　　q(x) ← 0;

繰り返し 高々 $(m-n+1)$ 回
($r(x)$ が 1次ずつ 小さくなる場合)

2. while $(\deg(r(x)) \geq \deg(u(x)))$ do
  $c \leftarrow lc(r(x)) / lc(u(x))$;   (除) 1回
  $d \leftarrow \deg(r(x)) - \deg(u(x))$;   (加) 1回
  $r(x) \leftarrow$ add-unipol-unipol (
    $r(x)$,
    $x^d \times$ mul-unipol-num (c, $u(x)$) $O(n)$
  );
  $q(x) \leftarrow q(x) + c \cdot x^d$;

3. return $(r(x), q(x))$. ☒

$O(\min\{d, n\})$ だが、実質上は $O(n)$ 項の係数を加えているため、$O(n)$.

合計 $\{O(n) + O(n)\} \times (m-n+1) + O(m) = O(mn)$
($m \geq n$ より.)

$O(m)$：擬除算の場合のみ.