平成11年度

卒業論文

C++による 常微分方程式の 全解探索アルゴリズムの構築について

研究指導：柏 木　雅 英　助教授

平成12年2月4日

早稲田大学 理工学部 情報学科

g93p048-9　　桜 井　幹 夫

目 次

1 序論 **1**

1.1 背景 2

1.2 本論文の目的 3

1.3 本論文の構成 3

2 区間演算 **4**

2.1 はじめに 5

2.2 区間演算の基本概念 5

2.3 区間演算に関する補題 8

2.4 この章のまとめ 9

3 常微分方程式の

全解探索アルゴリズム **10**

3.1 はじめに 11

3.2 $\phi(v, t_s, t_e)$ の計算 [1] 11

3.2.1 べき級数演算 11

3.2.2 Picard の反復法に基づく解の包み込みのアルゴリズム 13

3.3 Krawczyk 写像による微分方程式の解の存在検証法 [2] 14

3.3.1 Krawczyk 写像 15

3.3.2 解の存在・非存在の検証法 17
3.4 常微分方程式の全解探索アルゴリズム 17
3.5 おわりに 18

4 数値例 **19**

4.1 はじめに 20
4.2 数値検証の準備 20
4.2.1 プログラムの概要 20
4.2.2 数値検証の際の工夫点 21
4.3 数値例 21

5 まとめ **23**

謝辞 **25**

参考文献 **26**

付録 **27**

第 1 章

序論

1.1 背景

今日, コンピュータは, それ抜きでは社会が成り立たないほど, 日常の生活に深く浸透している. 技術の急速な進歩により, 従来研究機関にしかなかったような高度な CPU が普通の会社や, 個人の家庭にも存在するようになった.

その数値計算は通常有限桁の浮動小数点を用いた演算である. 人間社会では通常数値は 10 進数で表現されるが, コンピュータ内部は 2 進数または 16 進数などで数値を表現している. そのため,10 進数では 0.1 と表されるものが 2 進数のコンピュータ内部では 0.0001100110... の無限小数となる. 有限桁の浮動小数点では, ここで必ず丸め誤差が生じることになる. これは, 有限桁の浮動小数点演算で起こり得る誤差の一例であるが, その他演算における丸め誤差, 打ちきり誤差, 桁落ちなど様々な誤差がこの演算には潜んでいる.

こうした有限桁の浮動小数点演算は, 通常使われるような一般的な計算の場合で, 特に解に高い正確さ (計算機で出力される解が真解にどれだけ近いか) が必要でなければ十分である. しかしその演算による値には上で述べたような誤差が潜んでいるため, ある程度の正確さはあるとしても, どの程度の正確さあるいは誤差があるかは通常伺い知れない. 研究のデータのような厳密さを要求される場面でこの演算を用いることは, 研究結果の間違いといった危険をはらむことになる. また単純な計算の場合でも, 計算の繰り返しによる誤差の蓄積が, 間違った結果を導きだす場合もある. それは, 最近脚光を浴びているカオス理論と考えあわせるとその危険性が指摘できる. カオスという現象はある系において, 初期値に対する鋭敏な依存性として知られている. つまり初期値がわずかに違うだけでその後の軌道は, 全く違ったものになるということである. よって, 誤差を含んだ値による軌道は, 真の値による軌道とは全く違ったものになるが, 表面的にはそれが真の値によるものと誤認してしまうこともありうる. 以上より, 厳密さを要求される計算の場合だけでなく, 簡単な計算の繰り返しの場合でも誤差による影響は計り知れない.

様々な工学上の問題は, 線形または非線形の方程式で定式化され, 研究されてきている. 線形系の問題ではかなりの研究が進み, 今後は非線形系が重要なテーマになる. 非線形の解析は, 解析的に解く場合と, 解析的に解けない場合に数値解析によって解く場合が考えられる. 計算機が発達した現在では, 非線形系を数値解法を用いて解析を行うことは実用的にも可能であり, 有効な手段だといえる.

しかし, 計算機を用いた数値解析では, 上で述べたような誤差をいつも隣合わせであるといえる. そこで, 計算機での解と真解との誤差がどれほどであるかを正確に把握し保証することが重要であり, それを「 解を精度保証する 」という.

非線形方程式,

$$\frac{dx(t)}{dt} = f(x(t), t), \quad x(0) = a, \quad x(1) = b \tag{1.1}$$

の解 $x(t)$ を近似解から精度保証する方法についてはもうすでに研究されている. ただし,$x(t)$ は C^1 級とする. しかし, 十分よい近似解でなければ精度保証に失敗する. 例えば, 浮動小数点演算でニュートン法を用いて解を求める場合, 例え解が求められても, その演算を用いている以上誤差の混入があり値の保証はないので, 十分よい近似解といえるかわからない. さらに, ニュートン法で解を得るためには, 真解に十分近い初期点を与える必要がある. これは他の近似解を求める方法においても同様である.

こうしたことから, 全く解の存在が分からない時, 解の探索が必要となる. しかし, ホモトピー法など浮動小数点演算を用いた各種の解の探索法では, 解の探索においても解の精度保証においても万全とはいえないことが分かる. そこで, ある領域内の解を必ず見つけ, かつその解の精度を任意に保証することは, 計算機が更に発達して系を数値解析的に求めようという需要が増えるほど重要であるといえる. また, 非線形抵抗回路の最適な動作点を求める場合などでも, 全ての解を正確に求めたいという要求がある. そこで, ある領域内の全解を漏らさず精度保証付きで求めることは今後大切になると思われる.

1.2 本論文の目的

本論文は，2 点境界値問題の常微分方程式の全解探索のアルゴリズムを示す. そのためにまず、常微分方程式の初期値問題に対して近似解を与える関数 $\phi(v, t_s, t_e)$ の計算方法を示し, さらに Krawczyk 写像を用いて解が存在する領域を探索した時の解の存在・非存在の示し方を示す.

さらに, 探索領域の取り方によってこの 2 つの条件のいずれをも満たさなかった場合の対処法について示すことを目的とする.

1.3 本論文の構成

本論文の構成は次の通りである．

第 2 章では，本論文中で使用されている区間演算の概要について説明する.

第 3 章では，2 種類のベキ級数演算を用いた近似解を求めるアルゴリズムについて説明し, その近似解について Krawczyk 写像を用いた解の存在検証法について説明する. さらに, それらを使った全解探索アルゴリズムについて説明する.

第 4 章では，実際に第 3 章のアルゴリズムを使った全解探索の数値例を示す.

第 5 章では，結論として本論文の総括を行い，本研究に関する今後の課題についても述べる．

第 2 章

区間演算

2.1 はじめに

本論文では区間演算を使用している．本章では，その区間演算について説明する．

2.2 区間演算の基本概念

本節では，区間演算の基本概念についての説明を行う．R を実数全体の集合とすると，区間は次のように定義される．

定義 **2.2.1** (区間) $x, y \in R$ を端点に持つ区間 $[x, y]$ を

$$[x, y] = \{r \in R \mid x \leq y\} \tag{2.1}$$

で定義し，その集合を $I(R)$ で表す．また，区間 $[X, Y]$ の X, Y をそれぞれ下限，上限と呼ぶ．

ある区間 $X \in I(R)$ は， R の部分集合であり，無限集合である．また，ある実数 a が区間 X に内在しているとき， $a \in X$ と表す．

定義 **2.2.2** (等号) 区間 $X = [p, q]$ ， $Y = [v, w]$ に対して，等号を，

$$X = Y \quad \Longleftrightarrow \quad p = v \text{ かつ } q = w \tag{2.2}$$

で定義する．

定義 **2.2.3** (半順序) 区間 X, Y に対する半順序を， $\omega \in \{\leq, <, \geq, >\}$ として，

$$X \omega Y \quad \Longleftrightarrow \quad x \omega y \quad ({}^{\forall}x \in X, {}^{\forall}y \in Y) \tag{2.3}$$

で定義する．

また，ベクトルの半順序は次のように定義される．

定義 **2.2.4** (ベクトルの半順序) ベクトル $x, y \in R^n, x = (x_1, x_2, \cdots, x_n)^t, y = (y_1, y_2, \cdots, y_n)^t$ に対する半順序を， $\omega \in \{\leq, <, \geq, >\}$ として，

$$x \omega y \quad \Longleftrightarrow \quad x_i \omega y_i \quad ({}^{\forall}i) \tag{2.4}$$

で定義する．

2 つの区間 X, Y に対する二項演算が，次のように定義される．

定義 2.2.5 (二項演算) 区間 $X=[p,q], Y=[v,w]$ に対し，二項演算 $*\in\{+,-,\cdot,/\}$ を，

$$X*Y=\{x*y|x\in X, y\in Y\} \tag{2.5}$$

で定義する．ただし除算 $*=/$ の場合には $0\notin Y$ とする．

実際の区間演算は，

$$\begin{aligned} X+Y &= [p+v, q+w] && (2.6)\\ X-Y &= [p-v, q-w] && (2.7)\\ X\cdot Y &= [min\{pv,pw,qv,qw\}, max\{pv,pw,qv,qw\}] && (2.8)\\ X/Y &= [p,q]\cdot[1/w, 1/v] && (2.9) \end{aligned}$$

のように表現できる．

2 つの空間 X, Y に関する二項演算 $*$ の結果は再び区間となる．すなわち，区間全体は二項演算 $*$ に関して閉じている．区間に対するベクトル及び行列が，次のように定義される．

定義 2.2.6 (区間ベクトル) 区間ベクトル x を

$$x=(x_1,x_2,\cdots,x_n)^t \quad (x_1,x_2,\cdots,x_n\in I(R)) \tag{2.10}$$

で定義し，その集合を $I(R^n)$ で表す．

定義 2.2.7 (区間行列) 区間行列 A を,

$$A=(A_{11},\cdots,A_{mn}) \quad (A_{11},\cdots,A_{mn}\in I(R)) \tag{2.11}$$

で定義し，その集合を $I(R^{m\times n})$ で表す．

区間に対する集合演算として共通部分が，次のように定義される．

定義 2.2.8 (共通部分) 区間 $X=[p,q], Y=[v,w]$ に対して，包含関係を，

$$X\subset Y \iff v\le p \text{ かつ } q\le w \tag{2.12}$$

で定義する．同様にして，区間ベクトル $x=(x_1,x_2,\cdots,x_n)^t, y=(y_1,y_2,\cdots,y_n)^t$, 区間行列 $A=(A_{11},\cdots,A_{mn}), B=(B_{11},\cdots,B_{mn})$ に対して，

$$\begin{aligned} x\subset y &\iff x_i\subset y_i \quad ({}^\forall i) && (2.13)\\ A\subset B &\iff A_{ij}\subset B_{ij} \quad ({}^\forall i,j) && (2.14) \end{aligned}$$

が定義できる．

区間の中心及び半径は，次のように定義される．

定義 2.2.9 (区間の中心及び半径) 区間 $X = [p, q]$ の中心 $\mathrm{mid}(x)$, 半径 $\mathrm{rad}(x)$ を，

$$mid(X) = \frac{p+q}{2} \tag{2.15}$$

$$rad(X) = \frac{q-p}{2} \tag{2.16}$$

で定義する．

同様にして，区間ベクトル $x = (x_1, x_2, \cdots, x_n)^t$ ，区間行列 $A = (A_{11}, \cdots, A_{mn})$ に対して，

$$\begin{aligned} mid(x) &= (mid(x_1), mid(x_2), \cdots, mid(x_n))^t & (2.17)\\ rad(x) &= (rad(x_1), rad(x_2), \cdots, rad(x_n))^t & (2.18)\\ mid(A) &= (mid(A_{11}), \cdots, mid(A_{mn}))^t & (2.19)\\ rad(A) &= (rad(A_{11}), \cdots, rad(A_{mn}))^t & (2.20) \end{aligned}$$

が定義できる．

区間 X の中心と半径の関係は，

$$|x - mid(X)| \leq rad(X) \quad ({}^{\forall}x \in X) \tag{2.21}$$

となる．

式 (3.22) より， $mid(X)$ を区間 X 内に存在するある値 x^* の近似値と考えたとき，つまり，区間自身が誤差評価になっていることがわかる．区間の絶対値は，次のように定義される．

定義 2.2.10 (区間の絶対値) 区間 $X = [p, q]$ の絶対値 $|X|$ を,

$$|X| = max\{|p|, |q|\} \tag{2.22}$$

で定義する．

同様にして，区間ベクトル $x = (x_1, x_2, \cdots, x_n)^t$, 区間行列 $A = (A_{11}, \cdots, A_{mn})$ に対して，

$$\begin{aligned} |x| &= (|x_1|, \cdots, |x_n|)^t & (2.23)\\ |A| &= (|A_{11}|, \cdots, |A_{mn}|) & (2.24) \end{aligned}$$

で定義できる．

区間ベクトル及び区間行列のノルムは，最大値ノルムを一般化した次のノルムで定義される．

定義 2.2.11 (scaled maximum norm) 与えられたスケーリングベクトル $u = (u_1, u_2, \cdots, u_n)^t \in R^n, u > 0$ に対して，区間ベクトル $x \in I(R^n)$, 区間行列 $A \in I(R^{n\times n})$ のノルムを

$$\|x\|_u = max\{\frac{|x_i|}{u_i}| \ ^\forall i\} \tag{2.25}$$

$$\|A\|_u = \||A|_u\|_u \tag{2.26}$$

で定義する．

$u = (1, \cdots, 1)^t$ のときは通常の最大ノルムになる．

2.3 区間演算に関する補題

ここで，区間演算に関する補題を挙げておく．

補題 2.3.1 (最大値ノルムの規則) 区間ベクトルに $x, y \in I(R^n)$ に対して，

$$\|x\| = \||x|\| \geq 0 \tag{2.27}$$

$$\|ax\| = |a| \cdot \|x\| \quad (^\forall a \in R) \tag{2.28}$$

$$\|x\| - \|y\| \leq \|x \pm y\| \leq \|x\| + \|y\| \tag{2.29}$$

$$x \subset y \implies \|x\| \leq \|y\| \tag{2.30}$$

が成り立つ．同様に区間行列 $A, B \in I(R^{n\times n})$ に対して，

$$\|A\| = \||A|\| \geq 0 \tag{2.31}$$

$$\|Ax\| \leq \|a\| \cdot \|x\| \quad (^\forall x \in I(R^n)) \tag{2.32}$$

$$\|AB\| \leq \|A\| \cdot \|B\| \tag{2.33}$$

$$A \subset B \implies \|A\| \leq \|B\| \tag{2.34}$$

が成り立つ．これらは，$\|\cdot\| = \|\cdot\|_u$ としても成り立つ．

補題 2.3.2 (scaled maximum norm の規則) 区間ベクトル $x \in I(R^n)$, 区間行列 $A \in I(R^{n\times n})$，スケーリングベクトル $u \in R^n$，$u \geq 0$，そして $a \in R$ に対して，

$$\|x\|_u \leq a \iff |x| \leq au \tag{2.35}$$

$$\|x\|_u < a \iff |x| < au \tag{2.36}$$

$$\|A\|_u \leq a \iff |A|u \leq au \tag{2.37}$$

$$\|A\|_u < a \iff |A|u < au \tag{2.38}$$

が成り立つ．

2.4　この章のまとめ

本章では，区間演算の概説を行った．区間演算はその考え方が簡単で分かりやすいが，単純に区間演算を行っていくと区間の幅が大きく広がってしまい，その結果の値を使おうとしても非常に扱いにくくなってしまう．そこで，区間写像を利用して区間幅を縮小することで，扱い易くすると同時に精度保証も行うのである．

次の章では，常微分方程式の全解探索アルゴリズムについて説明する.

第 3 章

常微分方程式の

全解探索アルゴリズム

3.1 はじめに

本章では, 常微分方程式の全解探索アルゴリズムについて説明する. 探索する区間を初期値として, 第 3.2 節に示すべき級数演算によって近似解を得たあと, 第 3.3 節の Krawczyk 写像を用いた解の存在検証法に従って解の存在検証を行う, という手順で全解探索を行う.

3.2 $\phi(v, t_s, t_e)$ の計算 [1]

本節では, 常微分方程式の良い近似解を与えるためのアルゴリズムを示す.

まず, 常微分方程式 (1.1) の定義域を $0 = t_1 < t_2 < \cdots < t_m = 1$ のように分割して離散化し, $x(t_i)$ の近似 x_i を未知数とする有限次元方程式に帰着させて解くことにする. この過程でいわゆる離散化誤差が発生し, それを厳密に見積もることは極めて難しい. しかし, もし隣同士の値 x_i と x_{i+1} との関係を正確に記述することができれば, 真の値 (微分方程式の軌道上の値)$x(t_i)$ に一致する解 x_i を持つような有限次元方程式が得られるはずである. すなわち, 条件 $x(t_s) = v$ のもとで正確な $x(t_e)$ の値を与えるような関数 $\phi(v, t_s, t_e)$ を計算できれば, その正確な解を持つ有限次元方程式を,

$$\begin{aligned} x_2 &= \phi(x_1, t_1, t_2) \\ x_3 &= \phi(x_2, t_2, t_3) \\ &\vdots \\ x_m &= \phi(x_{m-1}, t_{m-1}, t_m) \end{aligned} \tag{3.1}$$

のように書くことが出来る. この方程式は $n \times m$ 個の未知数と $n \times (m-1)$ 個の方程式をもつ. したがって n 個の境界条件を付加することによって, 解けることが期待できる.

3.2.1 べき級数演算

本節では, 2 種類のベキ級数演算 (Power Series Arithmetic, 以下 PSA と略す) を定義する.

Type-I PSA Type-I PSA では, order-n のベキ級数を扱い, それより高次の項は切捨てる. $[a_0, a_1, a_2, \cdots a_n]$ という表記でベキ級数,

$$a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n \quad (+O(t^{n+1})) \tag{3.2}$$

を表すものとする. 加減算, 及び乗算は, 次のように定義する:

$$[a_0, \cdots, a_n] \pm [b_0, \cdots, b_n] = [a_0 \pm b_0, \cdots, a_n \pm b_n] \tag{3.3}$$

$$
\begin{aligned}
[a_0, \cdots, a_n] \times [b_0, \cdots, b_n] &= [c_0, \cdots, c_n], \\
c_k &= \sum_{i=0}^{k} a_i b_{k-i}
\end{aligned}
\tag{3.4}
$$

関数の適用は次のように行う:

$$
f([a_0, \cdots, a_n]) = f(a_0) + \sum_{i=1}^{n} \frac{1}{i!} f^{(i)}(a_0)[0, a_1, \cdots, a_n]^i \tag{3.5}
$$

上式中に現れる加算及び乗算は, 式 (3.3), 式 (3.4) によって行う.

除算は逆関数と乗算の組合せによって行う ($x/y = x \times (1/y)$).

Type-I PSA は, 全く打ち切りの無い演算を行った場合の最初の $(n+1)$ 項を保存するように定義された演算である.

Type-II PSA Type-II PSA でも order-n のベキ級数を扱う. Type-II PSA では取り扱う級数の定義域を $[0, d]$ のように定めている. $[a_0, a_1, a_2, \cdots, a_n]$ という表記でベキ級数,

$$
a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n \quad (+O(t^{n+1})) \tag{3.6}
$$

を表すことにする. このとき, 最後の項の係数 a_n は一般に区間となり, $[a_0, \cdots, a_n]$ は $f(t) \in a_0 + \cdots + a_n t^n$ in all t. を満たすような $[0, d]$ 上で定義された連続関数を表す.

加算と減算は Type-I PSA と同様に行う. また, 乗算は以下のように行う.

(1) $[a_0, \cdots, a_n]$ と $[b_0, \cdots, b_n]$ を打ちきりなしで乗算し, $2n$ 次の級数 $C = [c_0, \cdots, c_{2n}]$ を得る.

(2) C を n 次に減次する.

減次は以下のように行う.

定義 3.2.1 (減次=丸め) [1] $A = [a_0, \cdots, a_m]$ をベキ級数, $n(< m)$ は自然数とする. このとき A の次数 n への減次を次の $B = [b_0, \cdots, b_n]$ で定義する:

$$
b_i = a_i \quad (i \leq 0 \leq n-1) \tag{3.7}
$$

$$
b_n = a_n + \sum_{i=n+1}^{m} a_i [0, d]^{i-n} \tag{3.8}
$$

これは, 項 $a_i t^i$ を $a_i t^{i-n} t^n$ と変形し t^{i-n} を $[0, d]^{i-n}$ で置き換えることによって, 高次の剰余を t^n の係数に加えている. よって, 乗算の結果 C は丸めなしの乗算によって得られる可能性のある結果すべてを含んでいる.

関数の適用は次のように行う:

$$\begin{aligned} f([a_0,\cdots,a_n]) &= f(a_0)+\sum_{i=1}^{n}\frac{1}{i!}f^{(i)}(a_0)[0,a_1,\cdots,a_n]^i \\ &+ \frac{1}{n!}f^{(n)}(\sum_{i=0}^{n}a_i[0,d])[0,a_1,\cdots,a_n]^i] \end{aligned} \tag{3.9}$$

上式中に現れる加算及び乗算は Type-II PSA によって行う. この方法では Lagrange の剰余項を用いている. 除算は Type-I PSA と同様に行う.

3.2.2 Picard の反復法に基づく解の包み込みのアルゴリズム

式 (1.1) を同値な不動点形式:

$$x = v + \int_0^t f(x,t)dt \tag{3.10}$$

に変換し, 定数関数 v を初期値とする反復によって近似解を得, 真の解の存在を Schauder の不動点定理によって示す. アルゴリズムを示す.

アルゴリズム **3.2.1** [1] Type-II PSA の定義域を $[0,\Delta]$ とする.

Step-1 ベキ級数ベクトル X を

$$X = \begin{pmatrix} [v_1] \\ \vdots \\ [v_n] \end{pmatrix} \tag{3.11}$$

のように初期化し, $m=0$ とする.

Step-2(近似解の生成) Step-2(1)–Step-2(3) を適当な回数繰り返す.

Step-2(1) ベキ級数 T を, t を m 次で打ち切ったものとする.

$$T = \begin{cases} [0] & (m=0) \\ [0,1] & (m=1) \\ [0,1,0,\cdots,0] & (m>2) \end{cases} \tag{3.12}$$

Step-2(2) $X = f(X,T)$ を Type-I PSA で計算する. $X = v + \int_0^t X dt$ を計算する. これらによって X の次数は m から $m+1$ となる.

Step-2(3) $m = m+1$

Step-3 $T=[t_s,1,0,\cdots,0]$ (order-m) とする.

Step-4 $Y=f(X,T)$ を Type-II PSA で計算し, $Y=v+\int_0^t Y\,dt$ を計算し, 定義 3.2.1 によって Y の次数を. $m+1$ から m に減次する.

Step-5

$$r=\max_{1\le i\le n}|Y_i^{(m)}-X_i^{(m)}| \tag{3.13}$$

を計算する. ここで添字 $^{(k)}$ は t^k の係数を表す.

Step-6 Let $x_i^{(m)}=x_i^{(m)}+[-2r,2r]$ for $1\le i\le n$.

Step-7 $Y=f(X,T)$ を Type-II PSA で計算し, $Y=v+\int_0^t Y\,dt$ を計算し, 定義 3.2.1 によって Y の次数を. $m+1$ から m に減次する.

Step-8(存在検証) 全ての i について $Y_i^{(m)}\subset X_i^{(m)}$ ならば, Y に真解が存在することが Schauder の不動点定理により検証される.

Step-9(解の改良) Step-9(1)–Step-9(2) を $Y_i^{(m)}$ が十分小さくなるまで繰り返す.

Step-9(1) $X=f(Y,T)$ を Type-II PSA で計算し, $X=v+\int_0^t X\,dt$ を計算し, 定義 3.2.1 によって X の次数を. $m+1$ から m に減次する.

Step-9(2) 全ての i について $Y_i^{(m)}=Y_i^{(m)}\cap X_i^{(m)}$ とする.

Step-10 $\phi(v,t_s,t_e)$ は

$$\phi=\begin{pmatrix}\sum_{i=0}^m Y_1^{(i)}\Delta^i\\ \vdots\\ \sum_{i=0}^m Y_n^{(i)}\Delta^i\end{pmatrix}\text{で得られる.} \tag{3.14}$$

Step-8 において, $0\le k\le m-1$ で $Y_i^{(k)}=Y_i^{(k)}$ が常に成立し, よって最後の項だけで比較を行えばよい.

3.3 Krawczyk 写像による微分方程式の解の存在検証法 [2]

前節で近似解を求めるアルゴリズムを示した. しかし, その近似解はどの程度真解から離れているのかは分からない. そこで, その近似解 (区間で得られる) 内における解の存在・非存在の検証に本論文では Krawczyk 写像を用いている. 本節では, その Krawczyk 写像を使った解の存在検証法について説明する.

3.3.1 Krawczyk 写像

次の 1 階常微分方程式の非線形境界値問題について考える.

$$\begin{aligned} \frac{dx}{dt} &= f(x,t), \quad t \in J = [0,1] \\ g(x) &= 0 \end{aligned} \tag{3.15}$$

ただし, $f(x,t)$ は n 次元ベクトル値関数, g は境界条件を表す n 次元ベクトル値汎関数である. ここでは (3.15) 式の近似解 $c(t)$ が与えられたと仮定して, この近似解の近くに真解が存在するための十分条件を示す. 縮小写像の原理を用いて調べる.

$X = C[0,1;V]$ を区間 $J = [0,1]$ で連続な実 n 次元ベクトル値関数 $x(t) = (x_1, x_2, \cdots, x_n)$ の作る Banach 空間とする. scaled maximum norm を次のように与える:

$$\|x\|_u = \max_{t \in J} |x(t)|_u \tag{3.16}$$

ただし

$$|x|_u = \max_{1 \le i \le n} \frac{|x_i(t)|}{u_i} \tag{3.17}$$

ここで, $u = (u_1, u_2, \cdots, u_n)$ は定 n 次元ベクトルで正の成分を持つものとする. すなわち, $u_i > 0$ for $i = 1, 2, \cdots, n$. いま, $Y = X \times R^n$ を次のノルムを持つ Banach 空間とすると,

$$\|y\|_Y = max(\|u\|_u, \|e\|) \quad for\ y = (u,e) \in Y \tag{3.18}$$

$D = C^1[0,1;V]$ を連続な実 n 次元ベクトル値関数 $x(t) = (x_1, x_2, \cdots, x_n)$ の作る Banach 空間とする. さらに, $C[0,1;M]$ は区間 J 上で連続かつ微分可能な成分からなる $n \times n$ 行列値関数から作られる空間とする.

X を成分に持つ (3.15) 式の近似解 $c(t)$ が与えられているとする. ここで, 作用素 $F : D \subset X \to Y$ を次のように定義する:

$$Fx = (\frac{dx}{dt} - f(x,t), g(x)) \tag{3.19}$$

すると (3.15) 式は次の作用素方程式に書き換えられる:

$$Fx = 0 \tag{3.20}$$

以下では, $f : X \to X$, $g : X \to R^n$ が, J 上で連続で x に関して Fréchet 微分可能とする.

f の x に関する Jacobi 行列を $f_x(x,t)$ とし, g の Fréchet 微分を $g'(x)$ とする. すると D の要素について $F : X \to Y$ は Fréchet 微分可能であり, 次のように表される:

$$DF(x)h = (\frac{dh}{dt} - f_x(x,t)h, g'(x)h) \quad for\ h \in D \tag{3.21}$$

$$\tilde{L}h = (\frac{dh}{dt} - \tilde{A}(t)h, lh) \tag{3.22}$$

また $\tilde{\Phi}(t)$ を線形斉次系の基本解行列とする:

$$\frac{dz}{dt} = \tilde{A}(t)z \tag{3.23}$$

ただし $\tilde{\Phi}(0) = I$. いま, $\Phi(t) \in C^1[0,1;M]$ が $\tilde{\Phi}(0) = I$ を満たす $\tilde{\Phi}(t)$ の近似で, すべての $t \in J$ において逆行列をもつとする. ここで,

$$A(t) = \frac{d\Phi(t)}{dt}\Phi^{-1}(t) \tag{3.24}$$

とおくと次の関係式が成り立つ:

$$\frac{d\Phi(t)}{dt} = A(t)\Phi(t) \tag{3.25}$$

ただし $\Phi(-1) = I$. これは $\Phi(t)$ が

$$\frac{dz}{dt} = A(t)z \tag{3.26}$$

という線形斉次系における正確な基本解行列であることを意味する.

いま, 以下の作用素を定義する:

$$Lh = (\frac{dh}{dt} - A(t)h, lh) \quad for\ h \in D \tag{3.27}$$

このとき $G = l[\Phi(t)]$ とすると, 占部の定理により,G の逆行列が存在するとき L^{-1} が存在し次のように表せる:

$$L^{-1}(\phi, u) = H\phi + Su, \quad \phi \in X,\ u \in R^n \tag{3.28}$$

ただし H は X から $D \subset X$ への, また S は R^n から D への線形作用素でありそれぞれ次のように書ける:

$$H\phi = \Phi(t)\int_0^t \Phi^{-1}(s)\phi(s)ds - \Phi(t)G^{-1}l[\Phi(t)\int_0^t \Phi^{-1}(s)\phi(s)ds] \tag{3.29}$$

$$Su = \Phi(t)G^{-1}u \tag{3.30}$$

ここで, Newton 的作用素 $k : X \to X$ を導入する:

$$\begin{aligned} k(x) &= L^{-1}(L - F)x \\ &= L^{-1}(f(x,t) - A(t)x, l(x) - g(x)) \end{aligned} \tag{3.31}$$

$T(t)$ を $\mathrm{Mid}\ (T(t)) = c(t)$ を満たす区間関数とする. いま, 次の Krawczyk 作用素を定義する:

$$K(T) = k(c) + M(T - c) \tag{3.32}$$

ただし, $M = L^{-1}(L - DF(T))$ かつ $c = \mathrm{Mid}\ (T)$ である. さらに具体的に書くと,

$$\begin{aligned} M(T(t) - c) &= \Phi(t) \int_0^t \Phi^{-1}(s) R(s)(T(s) - c(s))ds \\ &- \Phi(t) G^{-1} l[\Phi(t) \int_0^t \Phi^{-1}(s) R(s)(T(s) - c(s))ds] \\ &- \Phi(t) G^{-1}(l - g'(T(t)))(T(t) - c(t)) \end{aligned} \tag{3.33}$$

ここで $R(t) = f_x(T(t), t) - A(t)$. このとき, 次の定理が成り立つ.

定理 3.3.1 [2] $T(t) \subset X$ を $\mathrm{Mid}\ (T(t)) = c(t)$ を満たす有界な区間関数とする. このとき次のことが成立する:

$$K(T(t)) \subset T(t) \tag{3.34}$$

$$\|M\|_u < 1 \tag{3.35}$$

上の $(3.34)(3.35)$ 式が共に成り立つとき, k の不動点 x^* が $T(t) \subset X$ 上に唯一存在する. さらに x^* は D 上の点であり $Fx^* = 0$ を満たし, かつ正則である. すなわち $DF(x^*)$ が逆行列をもつ.

3.3.2 解の存在 · 非存在の検証法

区間関数 $T(t)$ が式 (1.1) の近似解であり, 式 (3.34), (3.35) を満たすとき,$T(t)$ の中に真解が唯一つ存在する.

また, $T(t)$ について,

$$g(T(t)) \not\ni 0 \tag{3.36}$$

であれば, $T(t)$ の中に真解は存在しないといえる.

3.4 常微分方程式の全解探索アルゴリズム

以下にアルゴリズムを示す.

アルゴリズム 3.4.1 解の探索領域を $A \in R^n$ とする.

Step-0 $L = \{\mathbf{A}\}$ というリストを作成する.

Step-1 L の先頭から A を取り出す.

Step-2 $B = A + \alpha$ とする. ただし,

$$\alpha = [-\gamma \mathrm{rad}(A), \gamma \mathrm{rad}(A)] \quad (\gamma : \text{微小定数}) \tag{3.37}$$

Step-3 任意の $a \in B$ に対して, 以下の初期値問題の解を含む区間関数 $T(t)$ を求める.

$$\begin{aligned} \frac{dx}{dt} &= f(x,t) \\ x(0) &= a \end{aligned} \tag{3.38}$$

$g(T(t)) \not\ni 0$ ならば, 領域 A に対する解の非存在が証明される. Step-6 へ.

Step-4 前節で説明した方法により, $T(t)$ に対して解の存在検証を行う. 解の存在がいえれば, Step-6 へ.

Step-5 A を A_1 と A_2 に分割する. A_1 と A_2 をリスト L に追加する.

Step-6 リスト L が空ならば終了. そうでなければ,Step-1 へ.

3.5 おわりに

本章では, 常微分方程式の全解探索アルゴリズムについて述べた. 次章では実際に常微分方程式に対して全解探索を行っていく.

第 4 章

数値例

4.1 はじめに

本章では, 第 3 章で示した方法に従って実際に常微分方程式の全解探索を行う. 4.2 節では, 本章における数値検証を行うために作成したプログラムの概説を述べる. 4.3 節では以下に示す常微分方程式に対して全解探索を行った結果を示す.

$$\begin{aligned} \frac{d^2x}{dt^2} + 4x + 4(x - 2t + 1)^3 &= 8t - 3.6, \quad t \in [0, 1] \\ x(0) &= -0.9, \quad x(1) = 1.1 \end{aligned} \tag{4.1}$$

4.2 数値検証の準備

本節では, 本論文の数値検証のために作成したプログラムを概説し,(4.1) 式を解きやすくするための方法を示す. 第 3 章で示した全解探索アルゴリズムを実現するためには, 区間演算, べき級数演算が扱えるシステムが必要である.

例えば, 文献 [5] では基本ツールとして Calc を用い, 有理数演算を基本として区間演算, べき級数演算を実現していた. しかし, 常微分方程式の全解探索アルゴリズムのように多くの計算を伴う例では, 有理数演算のため, 多大な時間がかかってしまっていた.

そこで本論文では, プログラミング言語 C++の浮動小数点演算を用いて, これら区間演算, べき級数演算を実装し, ひいては上位の全解探索アルゴリズムを実装する.

4.2.1 プログラムの概要

ここでは, 数値検証のために私が作成したプログラムの特徴について簡単に述べる.

- 浮動小数点演算により精度保証を行う.
- プログラミング言語 C++を使用し, 区間, べき級数, べき級数のベクトル, 行列をクラスとして定義している. そして, それらを用いた演算式を演算子の多重定義により数学的な表記に近い形式で表現できるようにしている.
- 浮動小数点値の丸めには, 浮動小数点演算コプロセッサのアセンブリ言語を直接記述する方法を用いている. これにより, 区間演算を高速に実現している.
- 現在, 目的の常微分方程式に対応した最小限の構成であるが, クラスにそれぞれ演算子を追加するなどして, 容易に他の問題に対応することができる.
- 実行させると”ans.txt”ファイルに探索結果を出力する.

4.2.2 数値検証の際の工夫点

以下に式 (4.1) について全解探索を行う際の工夫点について述べる.

- 2 階の常微分方程式である式 (4.1) を 1 階連立常微分方程式に変えて考える. すなわち,

$$x_1 = x, \quad x_2 = \frac{dx}{dt} \tag{4.2}$$

として, 式 (4.1) を,

$$\begin{aligned} dx_1 &= x_2 \\ dx_2 &= -4x_1 - 4(x_1 - 2t + 1)^3 + 8t - 3.6 \end{aligned} \tag{4.3}$$

に直して計算を行う. また, 境界条件を表す作用素 g は以下のようになる.

$$g(x) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1(0) \\ x_2(0) \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1(1) \\ x_2(1) \end{pmatrix} - \begin{pmatrix} -0.9 \\ 1.1 \end{pmatrix} \tag{4.4}$$

- 式 (4.1) を $[0, 1]$ 区間で解くと, 精度保証がしにくくなるので, 変数変換を施し,$[0, 0.1]$ 区間になおして, 全解探索を行う. すなわち,

$$s = \frac{1}{10}t \tag{4.5}$$

として, 式 (4.3) を以下のように変形して, 計算を行う.

$$\begin{aligned} \frac{dx_1}{dt} &= 10x_2 \\ \frac{dx_2}{dt} &= 10(-4x_1 - 4(x_1 - 2 \cdot 10 \cdot s + 1)^3 + 8 \cdot 10 \cdot s - 3.6) \end{aligned} \tag{4.6}$$

4.3 数値例

本節では, 実際に式 (4.1) について, 全解探索を行なった結果を示す. 探索は区間 $[1.5, 2.5]$ で行う. 対象となる区間はそれぞれ外側に元の区間幅の 1.1 倍に拡大して検証している. プログラムを実行させると, 式 (4.1) は区間 $[1.99921875, 2.01640625]$ に一つ解を示した. プログラム全体の実行時間は, Pemtium II, 300MHz 上で, 6 分 23 秒であった. 文献 [5] の, 本論文と同程度のプログラムを同様に実行させた場合, 最低限 1 時間 20 分という見積もりとなる. このように,C++で実装することによって, 実行時間の大幅な短縮を得ることができた.

その解 $c(t)$ は次の通りである.

$$
\begin{aligned}
c_1(t) &= \texttt{[-0.900000000000000,-0.900000000000000]} \\
&+ \texttt{[1.999218750000000, 2.016406250000001]}\ t^1 \\
&+ \texttt{[-0.002000000000000,-0.002000000000000]}\ t^2 \\
&+ \texttt{[-0.011265625000000, 0.000536458333333]}\ t^3 \\
&+ \texttt{[0.000659750162760, 0.000687948404948]}\ t^4 \\
&+ \texttt{[-0.000111581114451, 0.002324698307037]}\ t^5 \\
&+ \texttt{[-0.000095352603082,-0.000075764224175]}\ t^6 \\
&+ \texttt{[-0.000230034291509, 0.000011898317703]}\ t^7 \\
&+ \texttt{[0.000001078955375, 0.000007292158979]}\ t^8 \\
&+ \texttt{[-0.000001149959628, 0.000013665103923]}\ t^9 \\
&+ \texttt{[-0.000000388673378, 0.000000830938689]}\ t^{10} \\
&+ \texttt{[-0.000000596037747, 0.000000197325815]}\ t^{11} \\
&+ \texttt{[-0.000000152185650, 0.000000020144281]}\ t^{12} \\
&+ \texttt{[-0.000000041755775, 0.000000026641730]}\ t^{13} \\
&+ \texttt{[-0.000000001487520, 0.000000018244190]}\ t^{14} \\
&+ \texttt{[-0.000000004710602, 0.000000008010518]}\ t^{15}
\end{aligned}
$$

$$
\begin{aligned}
c_2(t) &= \texttt{[1.999218750000000, 2.016406250000000]} \\
&+ \texttt{[-0.004000000000000,-0.004000000000000]}\ t^1 \\
&+ \texttt{[-0.033796875000001, 0.001609375000000]}\ t^2 \\
&+ \texttt{[0.002639000651041, 0.002751793619792]}\ t^3 \\
&+ \texttt{[-0.000557905572256, 0.011623491535187]}\ t^4 \\
&+ \texttt{[-0.000572115618490,-0.000454585345052]}\ t^5 \\
&+ \texttt{[-0.001610240040563, 0.000083288223919]}\ t^6 \\
&+ \texttt{[0.000008631642997, 0.000058337271833]}\ t^7 \\
&+ \texttt{[-0.000010349636649, 0.000122985935306]}\ t^8 \\
&+ \texttt{[-0.000003886733781, 0.000008309386893]}\ t^9 \\
&+ \texttt{[-0.000006556415222, 0.000002170583966]}\ t^{10} \\
&+ \texttt{[-0.000001826227805, 0.000000241731369]}\ t^{11} \\
&+ \texttt{[-0.000000542825073, 0.000000346342486]}\ t^{12} \\
&+ \texttt{[-0.000000020825282, 0.000000255418664]}\ t^{13} \\
&+ \texttt{[-0.000000026178614, 0.000000110705587]}\ t^{14} \\
&+ \texttt{[-0.000000048711767, 0.000000010986166]}\ t^{15}
\end{aligned}
$$

第 5 章

まとめ

本論文では, ベキ級数演算 (PSA) を用いた方法で, 近似解を求め, その近似解に対して Krawczyk 写像を用いて, 解の精度保証をするという方法で, 常微分方程式の解の全解探索を行った.

第 2 章では, 本論文全体の準備として区間演算の概説を行った. 間演算を用いると, 有限桁の数しか扱えない場合でも区間による計算結果の包み込みを行うことにより無限桁の数を扱うことが出来る. また丸め誤差などの数値計算における各種誤差の厳密な把握も可能になる.

第 3 章では,2 種類のベキ級数演算, さらにはそれらを用いた近似解を求めるアルゴリズムを示した. さらに, 精度保証の方法として,Krawczyk の区間写像について説明した. 区間演算はその考え方が簡単で分かりやすい反面, 単純に区間演算を行っていくと区間の幅が大きく広がってしまい, その結果が使いものにならなくなってしまうという問題があった. そこで Krawczyk の区間写像を用いることにより, 区間における真解の存在性を機械的に検証することを可能にしたのである. そして, 全解探索の際に使う区間分割について説明した. 最後に, 説明した手法を用いて, 全解探索を行うためのアルゴリズムを示した.

第 4 章では, まず数値検証を行うために作成したプログラムの概説を述べ, 第 3 章で説明した全解探索アルゴリズムを用いて, 式 (4.1) について全解探索を行い, その解を示した. さらに, その際に工夫した点についても示した.

謝辞

本研究は，著者が早稲田大学理工学部在学中に行われたものである．本研究を進めるにあたり，終始懇篤なる御指導，御鞭撻を賜り，著者の研究生活に常に暖かい御激励を頂きました指導教授 柏木雅英 助教授に深く感謝の意を表します．

また，著者の研究生活を通じて，機会ある毎に御指導，及び多くの御支援を賜りました大石進一 教授に深く感謝致します．

また，著者の学生生活を通じて御指導を賜りました情報学科の諸先生方に深く感謝致します．

また，良き先輩として機会ある毎に貴重な御助言を賜り，また，著者の長き学生生活を通じて御交流を持たせて頂いた，柏木研究室助手の相馬隆郎 氏に深く感謝致します．

また，良き同僚として常に多くの御助言並びに御助力を頂きました柏木研究室並びに大石研究室の先輩，同輩諸氏に感謝致します．

参考文献

[1] 柏木 雅英　大石 進一：“ベキ級数演算を用いた常微分方程式の精度保証付き数値計算”,
信学技報,TECHNICAL REPORT OF IEICE. NLP94-91 pp.65-70(1995-01).

[2] 大石 進一：“Two Topics in Nonlinear System Analysis through Fixed Point Theorems”,
IEICE TRANS.FUNDAMENTALS,VOL.E77-A,NO.7 pp.1144-1153(1994-07).

[3] 相馬 隆郎：“べき級数を用いたパラメータ依存常微分方程式の初期値問題の解法”,
電子通信基礎研究 · 夏期文献ゼミ資料, (1996).

[4] 神沢 雄智　中村 晴幸　柏木 雅英　大石 進一：“有限ステップで停止することが保証された非線形方程式の全ての解を精度保証付で求めるアルゴリズム”,
電子情報通信学会論文誌 A Vol.J79-A pp.1-7, (1996).

[5] 花井 誠：” 常微分方程式の全解探索アルゴリズムについて”,
早稲田大学理工学部 電子・情報通信学科 学士論文 (1998)

付録

```
/* ------------------------------------ <global.h> ---- */
#include "seclib.h"
#include "psa.h"
#include "mat.h"
#include "matx.h"

mat ode(mat func(mat&), mat ini, double start, double width, int order);

extern int KORDER_;
int Kraw(mat& T);
mat remake(mat& T);

extern int ORDER_;
void fullauto(Interval d, double ganma);

/* ------------------------------------ <main.cpp> ---- */
#include "global.h"

int ORDER_=35; /* 近似解の order */
int KORDER_=15; /* Krawczyk 写像の order */

void main ()
{
    cout << setiosflags(ios::scientific) << setprecision(15);

    double START_=1.5;              /* 探索区間の */
    double END_  =2.5;              /* 端点       */
    Interval d(START_,END_);
    fullauto(d, 0.1);
}

/*解く微分方程式の定義域を [0,0.1] に変換したもの*/
mat func(mat& x)
{
    mat y(2);
    taylor s(2, 0.0, 1.0);
```

```
    s=10*s.setorder(x[0].num-1);

    y[0]=10*x[1];
    y[1]=10*(-4*x[0] -4*((x[0]-2*s+1)^3)+8*s-3.6);
    return y;
}

/*解くべき微分方程式*/
mat orifunc(mat& x)
{
    mat y(2);
    taylor s(2, 0.0, 1.0);
    s=s.setorder(x[0].num-1);

    y[0]=x[1];
    y[1]=(-4*x[0] -4*((x[0]-2*s+1)^3)+8*s-3.6);
    return y;
}

/*f(x,t)のヤコビヤンを表す関数*/
matx fx_xt(mat& x)
{
    matx r(2);
    taylor s(2, 0.0, 1.0);
    s=s.setorder(x[0].num-1);

    r[0][0]=psa(1, 0.0);
    r[0][1]=psa(1, 1.0);
    r[1][0]=(-4-12*((x[0]-2*s+1)^2));
    r[1][1]=psa(1, 0.0);
    return r;
}

/*g(x)を定義する関数*/
mat g(mat& x)
{
    mat r(2);
    r[0]=x[0].eval(0)-(-0.9);
    r[1]=x[0].eval(1)-1.1;
    return r;
}

/*l(x)を定義*/ /* 2x2 */
matx l(matx& x)
{
    matx r(2);
```

```
    r[0][0]=x[0][0].eval(0);
    r[0][1]=x[0][1].eval(0);
    r[1][0]=x[0][0].eval(1);
    r[1][1]=x[0][1].eval(1);
    return r;
}

/*l(x)を定義*/ /* 2vec */
mat l(mat& x)
{
    mat r(2);
    r[0]=x[0].eval(0);
    r[1]=x[0].eval(1);
    return r;
}
```

```
/* ------------------------------------ <allsol.cpp> ---- */
#include <stdlib.h>
#include <fstream.h>
#include "global.h"

#define L_max 50
#define A_max 10

int L_data;
int A_data;
Interval L[L_max];
matx A(A_max, 2);

Interval AI[A_max];
double min_width;
void resultout();
mat func(mat& x);

/* x = 1 における境界条件を判定する関数*/
int check(mat& x)
{
    Interval e;
    e=x[0].eval(0.1).array[0];
    cout << "区間は";
    cout << e << endl;

    if((e.left<=1.1)&&(e.right>=1.1))
        return 1;       /*境界条件満たす*/
    else
        return (-1);    /*境界条件満たさず*/
}

/*全解探索アルゴリズムを実行する関数*/
void fullauto(Interval d, double ganma)
{
    int i;
    mat init(2), x(2);
    min_width=d.width();

    for(i=0;i<L_max;i++)
        L[i]=Interval(0,0);
    for(i=0;i<A_max;i++)
        for(int j=0;j<2;j++)
            A[i][j]=psa(1, 0.0);

    /*L を初期化*/
```

```
L[0]=d;
L_data=1;
A_data=0;

/*L が空になるまで探索する*/
while(L[0]!=Interval(0,0))
{
    /*初期値を設定*/
    Interval start=L[0];
    double width=L[0].width()/2;
    init[0]=psa(1, -0.9);
    init[1]=L[0]+Interval(-ganma*width, ganma*width);

    for(i=0;i<L_data;i++)
        L[i]=L[i+1];

    L_data=L_data-1;

    /*近似解を求める*/
    cout << "初期値は" << endl;
    cout << init ;
    x=ode(func,init,0,0.1,ORDER_);

    /*近似解の包み込みに失敗した時の処理*/
    if(x.num==0){
        cout << "包み込み失敗 -> 区間分割" << endl;

        for(i=L_data;i>=0;i--)
            L[i+2]=L[i];

        /*区間分割*/
        L[0].left=start.left;
        L[0].right=start.left+start.width()/2;
        L[1].left=start.left+start.width()/2;
        L[1].right=start.right;
        L_data=L_data+2;
        for(i=0;i<L_data;i++){
            cout << "L[" << i << "]=";
            cout << L[i] << endl;
        }
    }else{
        /*近似解の包み込みに成功した時の処理*/
        cout << "包み込み成功" << endl;

        /*x = 1 における境界条件の判定*/
        cout << "x=1 での境界条件" << endl;
```

```
    int s=check(x);

    if(s==1){
        cout << "境界条件満たす" << endl;
        int KT=Kraw(x);

        if(KT==1){
            /*解の存在確認*/
            cout << "解が存在" << endl;
            A[A_data]=x;
            AI[A_data]=init[1].array[0];
            A_data++;
            double width=init[1].array[0].width();
            if(min_width>width)
                min_width=width;
        }
        else if(KT==0){
            /*不定・区間分割*/
            cout << "不定 -> 区間分割" << endl;

            for(i=L_data;i>=0;i--)
                L[i+2]=L[i];

            L[0].left=start.left;
            L[0].right=start.left+start.width()/2;
            L[1].left=start.left+start.width()/2;
            L[1].right=start.right;
            L_data=L_data+2;
            for(i=0;i<L_data;i++){
                cout << "L[" << i << "]=";
                cout << L[i] << endl;
            }
        }
        else if(KT==(-1)){
            /*解の非存在確認*/
            cout << "非存在確認" << endl;
            double width=init[1].array[0].width();
            if(min_width>width)
                min_width=width;
        }else
            cout << "エラーです" << endl;
    }
    /*境界条件を満たさなかった時*/
    else
        cout << "境界条件不適" << endl;
}
```

```
        cout << endl;
    }
    resultout();
}

void resultout()
{
    int i;
    ofstream of;
    of.open("ans.txt", ios::out);
    of << setiosflags(ios::scientific) << setprecision(15);

    of << "探索結果 :" << endl;
    for(i=0;i<A_data;i++){
        of << "区間 : " << AI[i] << endl;
        of << A[i] << endl;
    }
    of << "最小幅 : " << min_width << endl;
    of.close();
}
```

```
/* ------------------------------------ <kraw.cpp> ---- */
#include <stdlib.h>
#include <math.h>
#include <assert.h>
#include "global.h"

mat orifunc(mat& x);
matx fx_xt(mat& x);
mat g(mat& x);

mat mid(mat& T);
matx _phi();
mat Atirda(mat& x);
matx inv_phi();
mat phi_1(mat& x);
matx l(matx& x);
mat l(mat& x);
mat invL(mat x,mat y);
mat H(mat& x);
mat S(mat& x);
mat MT_c(mat& T,mat& ct);
int contain_check(psa x, psa y);

mat ct(0);
matx phi(0), At(0), AtT(0), invphi(0), invG(0), Rt(0);

int Kraw(mat& T)
{
    int i;
    mat T1(T.num);

/*******[0,0.1] 区間を [0,1] 区間に広げ, 近似解の中央をとる*********/
    T1=T;
    T=remake(T);
    ct=mid(T);

/*******Φを求める*********************************************/
    phi=_phi();

/*******A(t) を求める*****************************************/
    At=fx_xt(ct);

/*******Φ-1 を求める*****************************************/
    invphi=inv_phi();

/*******G-1 を求める*****************************************/
```

```
    matx G(0);
    G=l(phi);
    invG=inv(G);

/*******R(t)の計算*****************************************/
    Rt=fx_xt(T)-At;

/*******Krawczyk作用素**************************************/
    mat dct(T.num);
    for(i=0;i<T.num;i++){
        dct[i]=ct[i].dif();
        dct[i]=dct[i].setorder(KORDER_);
    }

    mat L_F(0);
    L_F=invL(dct-orifunc(ct),g(ct));
    mat MTc(0);
    MTc=MT_c(T,ct);
    mat W(0);
    W=-L_F+MTc;

/*******包含関係*******************************************/
    int* r=(int*)malloc(sizeof(int)*T.num);

    for(i=0; i<T.num; i++)
        r[i]=contain_check(W[i],T[i]-ct[i]);

    int ret=1;
    for(i=0; i<T.num; i++){
        if(r[i]==-1){
            ret=-1;
            break;
        }
        if(r[i]==0) ret=0;
    }
    free(r);
    return ret;
}

/* [0,0.1]区間を[0,1]区間に広げる */
mat remake(mat& T)
{
    mat r(T.num);
    psa r0(T[0].num, ARG_TERM);

    for(int i=0; i<T.num; i++){
```

```
        for(int j=0;j<T[0].num;j++)
            r0.array[j]=pow(0.1, j)*T[i].array[j];
        r[i]=r0;
    }
    return r;
}

/*区間を含むベキ級数の中心を取る関数*/
mat mid(mat& T)
{
    mat r(T.num);
    psa r0(T[0].num, ARG_TERM);

    for(int i=0; i<T.num; i++){
        for(int j=0; j<T[i].num; j++)
            r0.array[j]=Mid(T[i].array[j]);
        r[i]=r0;
    }
    return r;
}

/* Φを求める */ /* 2x2 */
matx _phi()
{
    mat E0(2), E1(2);
    matx r(2);
    E0[0]=psa(1, 1.0);
    E0[1]=psa(1, 0.0);
    E1[0]=psa(1, 0.0);
    E1[1]=psa(1, 1.0);

    mat phi0(2), phi1(2);
    phi0=ode(Atirda, E0, 0, 0.1, KORDER_);
    phi1=ode(Atirda, E1, 0, 0.1, KORDER_);

    r[0][0]=phi0[0];
    r[0][1]=phi1[0];
    r[1][0]=phi0[1];
    r[1][1]=phi1[1];
    return r;
}

/*A~(t)=f(ct,t)*xと近似*/
mat Atirda(mat& x)
{
    mat r(0);
```

```
    r=fx_xt(ct)*x;
    return r;
}

/*Φ-1を求めるプログラム*/ /* 2x2 */
matx inv_phi()
{
    AtT=tenchi(At);

    mat E0(2), E1(2);
    E0[0]=psa(1, 1.0);
    E0[1]=psa(1, 0.0);
    E1[0]=psa(1, 0.0);
    E1[1]=psa(1, 1.0);

    matx r(2);
    mat r0(2), r1(2);
    r0=ode(phi_1, E0, 0, 0.1, KORDER_);/*２つの縦ベクトルに分けて計算する*/
    r1=ode(phi_1, E1, 0, 0.1, KORDER_);

    r[0][0]=r0[0];
    r[0][1]=r1[0];
    r[1][0]=r0[1];
    r[1][1]=r1[1];

    matx rT(2);
    rT=tenchi(r);
    return rT;
}

/*Φ-1を求めるための微分方程式*/
mat phi_1(mat& x)
{
    mat r(0);
    r=-AtT*x;
    return r;
}

/*L-1(φ,v)=Hφ+Sv*/
mat invL(mat x,mat y)
{
    mat r(0);
    r=H(x)+S(y);
    return r;
}
```

```
/*Hφを定義*/
mat H(mat& x)
{
    mat p(0);
    p=invphi*x; /*被積分関数を計算*/

    mat integ(x.num);
    /*積分部分の計算*/
    for(int i=0;i<x.num;i++){
        integ[i]=p[i].integrate();
        integ[i]=integ[i].setorder(KORDER_);
        integ[i]=integ[i]-integ[i].eval(0).setorder(KORDER_);
    }

    mat phiint(0);
    phiint=phi*integ;

    mat r(0);
    r=phiint-phi*invG*l(phiint);
    return r;
}

/*Svの定義*/
mat S(mat& x)
{
    mat r(0);
    r=phi*invG*x;
    return r;
}

/*M(T-c)の定義*/
mat MT_c(mat& T,mat& ct)
{
    assert(T.num==ct.num);
    mat p(0);
    p=invphi*Rt*(T-ct);     /*被積分関数を先に計算*/

    mat integ(T.num);
    /*積分計算*/
    for(int i=0;i<T.num;i++){
        integ[i]=p[i].integrate();
        integ[i]=integ[i].setorder(KORDER_);
        integ[i]=integ[i]-integ[i].eval(0).setorder(KORDER_);
    }
    mat phiint(0);
    phiint=phi*integ;
```

```
    mat r(0);
    r=phiint-phi*invG*l(phiint);
    return r;
}

/*２つの区間の包含関係を調べる*/
int contain(psa& xp, psa& yp)
{
    /*入力が両方とも区間である時の処理*/
    Interval x, y;
    x=xp.array[0];
    y=yp.array[0];
    if((y.left <= x.left) && (x.right <= y.right))
        return 1;
    else if((y.left > x.right) || (x.left > y.right))
        return (-1);
    else
        return 0;
}

/*２つのベキ級数の包含関係を調べる*/
int contain_check(psa x, psa y)
{
    int f=0;
    /*[0,1]区間を0.1間隔に切って，それぞれの部分について包含関係を調べる*/
    for(int i=0;i<((1/0.1)+1);i++){
        psa xi=x.eval(0.1*i);
        psa yi=y.eval(0.1*i);
        int c=contain(xi,yi);
        if(c==(-1))
            return (-1);    /*解の非存在が言えたら（-1）を返す*/
        else
            f+=c;
    }
    if(f==11)
        return 1;       /*解の存在が言えたら１を返す*/
    else
        return 0;       /*存在・非存在が断定できない時０を返す*/
}
```

```
/* ------------------------------------ <ode.cpp> ---- */
#include <stdlib.h>
#include "global.h"

int ode_check(CSec& y, CSec& x);

mat ode(mat func(mat&), mat init, double start, double width, int order)
{
    int n = init.num;
    int i, j;
    psa_width=0;
    mat x(n);

    /* step 1 */
    for(i=0; i<n; i++) x[i] = init[i];

    /* step 2 */
    for(j=0; j<order; j++){
        x = func(x);
        for(i=0; i<n; i++){
            x[i] = x[i].integrate();
            init[i]=init[i].setorder(init[i].num);
            x[i] = x[i].setorder(init[i].num-1)+init[i];
        }
    }
    psa_width=width;

    /* step 4 */
    mat y(n);
    y = func(x);
    for(i=0; i<n; i++){
        y[i] = y[i].integrate().setorder(order);
        y[i] = y[i]+init[i];
    }

    /* step 5 */
    double temp = 0.0;
    for(i=0; i<n; i++)
        temp = __max(temp, Abs(y[i].array[order] - x[i].array[order]));

    /* step 6 */
    for(i=0; i<n; i++)
        x[i].array[order] += temp * Interval(-2.0, 2.0);

    /* step 7 */
    y = func(x);
```

```
    for(i=0; i<n; i++){
        y[i] = y[i].integrate().setorder(order);
        y[i] = y[i]+init[i];
    }

    /* step 8 */
    temp = 1; /* true */
    for(i=0; i<n; i++)
        temp = temp && Iscontained(y[i].array[order], x[i].array[order]);
    if(!temp){
        /*cout << "can't find solution" << endl;*/
        return 0;
    }

    /* step 9 */
    for(j=0; 1; j++){
        x = func(y);
        for(i=0; i<n; i++){
            x[i] = x[i].integrate().setorder(order);
            x[i] = x[i] + init[i];
        }
        temp = 0; /* false */
        for(i=0; i<n; i++)
            temp = ode_check(y[i].array[order], x[i].array[order]) || temp;
        if(!temp) break;
    }
    return y;
}

int ode_check(CSec& y, CSec& x)
{
    int f = 0; /* false */
    if(x.left > y.left){
        y.left = x.left;
        f = 1;
    }
    if(x.right < y.right){
        y.right = x.right;
        f = 1;
    }
    if(y.left > y.right){
        cout << "something wrong" << endl;
        exit(0);
    }
    return f;
}
```

```
/* ------------------------------------ <seclib.h> ---- */
#include <iomanip.h>

extern int _RoundNear;
extern int _RoundDown;
extern int _RoundUp;

#ifdef _MSC_VER
#define RoundNear() __asm {__asm fldcw word ptr ds:OFFSET _RoundNear}
#define RoundDown() __asm {__asm fldcw word ptr ds:OFFSET _RoundDown}
#define RoundUp()   __asm {__asm fldcw word ptr ds:OFFSET _RoundUp}
#else
#define __max(a,b)  (((a) > (b)) ? (a) : (b))
#define __min(a,b)  (((a) < (b)) ? (a) : (b))
#define RoundNear() asm volatile ("fldcw __RoundNear")
#define RoundDown() asm volatile ("fldcw __RoundDown")
#define RoundUp()   asm volatile ("fldcw __RoundUp")
#endif

class ostream;
class istream;

typedef class CSec
{
public:
    double left;
    double right;

    CSec();
    CSec(double a);
    CSec(double a, double b);

    double width();
    CSec& operator = (const CSec&);
    CSec& operator = (const double&);
    CSec& operator += (const CSec&);

    friend int operator != (const CSec&, const CSec&);
    friend CSec operator + (const CSec&, const CSec&);
    friend CSec operator + (const double&, const CSec&);
    friend CSec operator + (const CSec&, const double&);

    friend CSec operator - (const CSec&, const CSec&);
    friend CSec operator - (const double&, const CSec&);
    friend CSec operator - (const CSec&, const double&);
```

```
    friend CSec operator * (const CSec&, const CSec&);
    friend CSec operator * (const double&, const CSec&);
    friend CSec operator * (const CSec&, const double&);

    friend CSec operator / (const CSec&, const CSec&);
    friend CSec operator / (const double&, const CSec&);
    friend CSec operator / (const CSec&, const double&);

    friend CSec operator ^ (const CSec&, const int&);
    friend CSec Pow(const CSec&, const int&);
    friend double Abs(const CSec&);
    friend double Mid(const CSec&);
    friend int Iscontained(const CSec&, const CSec&);
    friend ostream& operator << (ostream&, const CSec&);
    friend istream& operator >> (istream&, CSec&);
} Interval;

inline CSec operator + (const double& a, const CSec& b)
{
    return CSec(a)+b;
}

inline CSec operator + (const CSec& a, const double& b)
{
    return a+CSec(b);
}

inline CSec operator - (const double& a, const CSec& b)
{
    return CSec(a)-b;
}

inline CSec operator - (const CSec& a, const double& b)
{
    return a-CSec(b);
}

inline CSec operator * (const double& a, const CSec& b)
{
    return CSec(a)*b;
}

inline CSec operator * (const CSec& a, const double& b)
{
    return a*CSec(b);
}
```

```
inline CSec operator / (const double& a, const CSec& b)
{
    return CSec(a)/b;
}

inline CSec operator / (const CSec& a, const double& b)
{
    return a/CSec(b);
}

inline CSec operator ^ (const CSec& a, const int& b )
{
    return Pow(a, b);
}

/* ------------------------------------ <seclib.cpp> ---- */
#include <stdlib.h>
#include <math.h>
#include <assert.h>
#include "seclib.h"

int _RoundNear = 0x133a;
int _RoundDown = 0x173a;
int _RoundUp   = 0x1b3a;

CSec::CSec()
{
    left=right=0;
}

CSec::CSec(double a)
{
    left=right=a;
}

CSec::CSec(double a, double b)
{
    if(a<=b){
        left=a; right=b;
    }else{
        left=b; right=a;
    }
}

double CSec::width()
{
```

```
    assert(left<=right);
    return right - left;
}

CSec& CSec::operator = (const CSec& a)
{
    left=a.left;
    right=a.right;
    return  *this;
}

CSec& CSec::operator = (const double& a)
{
    left=right=a;
    return  *this;
}

CSec& CSec::operator += (const CSec& a)
{
    *this=*this+a;
    return *this;
}

int operator != (const CSec& a, const CSec& b)
{
    if(a.left==b.left && a.right==b.right)
        return 0;
    else
        return 1;
}

CSec operator + (const CSec& a, const CSec& b)
{
    assert(a.left<=a.right);
    assert(b.left<=b.right);

    CSec c;
    RoundDown();
    c.left=a.left+b.left;
    RoundUp();
    c.right=a.right+b.right;
    RoundNear();
    assert(c.left<=c.right);
    return c;
}
```

```
CSec operator - (const CSec& a, const CSec& b)
{
    assert(a.left<=a.right);
    assert(b.left<=b.right);

    CSec c;
    RoundDown();
    c.left=a.left-b.right;
    RoundUp();
    c.right=a.right-b.left;
    RoundNear();
    assert(c.left<=c.right);
    return c;
}

CSec operator * (const CSec& a, const CSec& b)
{
    assert(a.left<=a.right);
    assert(b.left<=b.right);

    int x,y;
    if(a.left*a.right>0){
        if(a.right<0) x=0; else x=2;
    }else x=1;
    if(b.left*b.right>0){
        if(b.right<0) y=0; else y=2;
    }else y=1;

    CSec c;
    switch(y*3+x){
        case 0:
            RoundDown();
            c.left=a.right*b.right;
            RoundUp();
            c.right=a.left*b.left;
            break;
        case 1:
            RoundDown();
            c.left=a.right*b.left;
            RoundUp();
            c.right=a.left*b.left;
            break;
        case 2:
            RoundDown();
            c.left=a.right*b.left;
            RoundUp();
```

```
            c.right=a.left*b.right;
            break;
        case 3:
            RoundDown();
            c.left=a.left*b.right;
            RoundUp();
            c.right=a.left*b.left;
            break;
        case 4:
            RoundDown();
            c.left=__min(a.left*b.right, a.right*b.left);
            RoundUp();
            c.right=__max(a.left*b.left, a.right*b.right);
            break;
        case 5:
            RoundDown();
            c.left=a.right*b.left;
            RoundUp();
            c.right=a.right*b.right;
            break;
        case 6:
            RoundDown();
            c.left=a.left*b.right;
            RoundUp();
            c.right=a.right*b.left;
            break;
        case 7:
            RoundDown();
            c.left=a.left*b.right;
            RoundUp();
            c.right=a.right*b.right;
            break;
        case 8:
            RoundDown();
            c.left=a.left*b.left;
            RoundUp();
            c.right=a.right*b.right;
            break;
    }
    RoundNear();
    assert(c.left<=c.right);
    return c;
}

CSec operator / (const CSec& a, const CSec& b)
{
```

```
assert(a.left<=a.right);
assert(b.left<=b.right);

int x,y;
if(a.left*a.right>0){
    if(a.right<0) x=0; else x=2;
}else x=1;

if(b.left*b.right>0){
    if(b.right<0) y=0; else y=1;
}else{
    cout << "Assert: Zero division!" << endl;
    exit(0);
}

CSec c;
switch(y*3+x){
    case 0:
        RoundDown();
        c.left=a.right/b.left;
        RoundUp();
        c.right=a.left/b.right;
        break;
    case 1:
        RoundDown();
        c.left=a.right/b.right;
        RoundUp();
        c.right=a.left/b.right;
        break;
    case 2:
        RoundDown();
        c.left=a.right/b.right;
        RoundUp();
        c.right=a.left/b.left;
        break;
    case 3:
        RoundDown();
        c.left=a.left/b.left;
        RoundUp();
        c.right=a.right/b.right;
        break;
    case 4:
        RoundDown();
        c.left=a.left/b.left;
        RoundUp();
        c.right=a.right/b.left;
```

```
            break;
        case 5:
            RoundDown();
            c.left=a.left/b.right;
            RoundUp();
            c.right=a.right/b.left;
            break;
    }
    RoundNear();
    assert(c.left<=c.right);
    return c;
}

CSec Pow(const CSec& a, const int& b)
{
    CSec c=1.0;
    for(int i=0;i<b; i++)
        c=c*a;
    return c;
}

double Abs(const CSec& a)
{
    return __max(fabs(a.left), fabs(a.right));
}

double Mid(const CSec& a)
{
    return (a.left + a.right) / 2;
}

int Iscontained(const CSec& a, const CSec& b)
{
    return (b.left <= a.left) && (a.right <= b.right);
}

ostream& operator << (ostream& s, const CSec& a)
{
    return s << "[" << a.left << "," << a.right << "]" ;
}

istream& operator >> (istream& s, CSec& a)
{
    return s >> a.left >> a.right;
}
```

```
/* ------------------------------------ <psa.h> ---- */
#define ARG_TERM -1.0e+300

extern double psa_width;

class psa
{
public:
    int num;
    Interval* array;

    psa();
    psa(int m, ...);
    psa(const psa& cp);
    virtual ~psa();
    void clean();
    void copy(const psa& cp);

    psa integrate(); /* ベキ級数の積分 */
    psa dif(); /* ベキ級数の微分 */
    psa setorder(int j);
    psa eval(double t);
    Interval range();

    psa& operator = (const psa& cp);
    psa& operator = (const CSec& cp);

    friend psa operator + (const psa&, const psa&);
    friend psa operator + (const double&, const psa&);
    friend psa operator + (const psa&, const double&);

    friend psa operator - (const psa&, const psa&);
    friend psa operator - (const double&, const psa&);
    friend psa operator - (const psa&, const double&);

    friend psa operator * (const psa&, const psa&);
    friend psa operator * (const double&, const psa&);
    friend psa operator * (const psa&, const double&);

    friend psa operator / (const psa& a, const psa&);
    friend psa operator / (const double&, const psa&);
    friend psa operator / (const psa&, const double&);

    friend psa operator ^ (const psa&, const int&);
    friend psa Pow(psa x, int y);
    friend psa Inv(psa x);
```

```
    friend ostream& operator << (ostream&, const psa&);
};
typedef psa taylor;
typedef psa *ppsa;

/* ------------------------------------ <psa.cpp> ---- */
#include <stdlib.h>
#include <stdarg.h>
#include <math.h>
#include <assert.h>
#include "seclib.h"
#include "psa.h"

double psa_width=1;

psa::psa()
{
    cout << "psa: Don't use default!" << endl;
    exit(0);
}

psa::psa(int m, ...)
{
    num=m;
    array=(Interval*)malloc(sizeof(Interval)*num);

    va_list marker;
    va_start( marker, m );
    double arg;

    arg = va_arg( marker, double);
    for(int i=0; i<m; i++){
        if(arg!=ARG_TERM){
            array[i] = arg;
            arg = va_arg( marker, double);
        }else
            array[i] = 0.0;
    }
    va_end( marker );
}

psa::psa(const psa& cp)
{
    num=0;
    copy(cp);
}
```

```
psa::~psa()
{
    clean();
}

void psa::clean()
{
    if(array!=0) free(array);
}

void psa::copy(const psa& cp)
{
    num=cp.num;
    if(cp.array!=0){
        array=(Interval*)malloc(sizeof(Interval)*num);
        for(int i=0; i<num; i++)
            array[i]=cp.array[i];
    }else
        array=0;
}

psa psa::integrate() /* ベキ級数の積分 */
{
    psa ret(num+1, ARG_TERM);
    ret.array[0] = 0.0;
    for(int i=0; i<num; i++)
        ret.array[i+1] = array[i]/(i+1);
    return ret;
}

psa psa::dif() /* ベキ級数の微分 */
{
    psa r(num-1, ARG_TERM);
    for(int i=0; i<num-1; i++)
        r.array[i]=(i+1)*array[i+1];
    return r;
}

psa psa::setorder(int n)
{
    psa ret(n+1, ARG_TERM);
    int s = __min(n+1, num);

    for(int i=0; i<s; i++) ret.array[i] = array[i];
    if(n+1 >= num){
        for (int i=s; i<n+1; i++) ret.array[i] = 0.0;
```

```
        return ret;
    }

    if(psa_width){
        Interval T(0, psa_width);
        Interval z = 0.0;
        int s1 = num - 1;
        for(int i=s1; i>=s; i--) z = z * T + array[i];
        ret.array[s-1] += z * T;
    }
    return ret;
}

psa psa::eval(double t)
{
    if(psa_width && (t<0 || t>psa_width)){
        /*cout << "taylor_eval : seem to be out of range" << endl;*/
    }

    /* Horner's method */
    Interval r = array[num-1];
    int s = num - 2;
    for(int i=s; i>=0; i--) r = r * t + array[i];

    psa ret(1, 0.0);
    ret.array[0]=r;
    return ret;
}

Interval psa::range()
{
    if(psa_width==0){
        cout << "can't calculate range ... set taylor_width" << endl;
        exit(0);
    }

    Interval T(0, psa_width);
    Interval r = array[num-1];
    int s = num - 2;
    for(int i=s; i>=0; i--) r = r * T + array[i];
    return r;
}

psa& psa::operator = (const psa& cp)
{
    clean();
```

```
    copy(cp);
    return *this;
}

psa& psa::operator = (const CSec& d)
{
    assert(num==1);
    clean();
    array=(Interval*)malloc(sizeof(Interval)*num);
    array[0]=d;
    return *this;
}

psa operator + (const psa& a, const psa& b)
{
    assert(a.array!=0 && b.array!=0);
    assert(a.num==b.num);

    psa ret(a.num, ARG_TERM);
    for(int i=0; i<a.num; i++)
        ret.array[i]=a.array[i]+b.array[i];
    return ret;
}

psa operator + (const double& a, const psa& b)
{
    return psa(b.num, a, ARG_TERM)+b;
}

psa operator + (const psa& a, const double& b)
{
    return a+psa(a.num, b, ARG_TERM);
}

psa operator - (const psa& a, const psa& b)
{
    assert(a.array!=0 && b.array!=0);
    assert(a.num==b.num);

    psa ret(a.num, ARG_TERM);

    for(int i=0; i<a.num; i++)
        ret.array[i]=a.array[i]-b.array[i];
    return ret;
}
```

```
psa operator - (const double& a, const psa& b)
{
    return psa(b.num, a, ARG_TERM)-b;
}

psa operator - (const psa& a, const double& b)
{
    return a-psa(a.num, b, ARG_TERM);
}

psa operator * (const psa& a, const psa& b)
{
    assert(a.num==b.num);
    assert(a.array!=0 && b.array!=0);
    psa ret(a.num, ARG_TERM);

    for(int k=0; k<a.num; k++){
        ret.array[k]=0.0;
        for(int i=0; i<=k; i++)
            ret.array[k]+=a.array[i]*b.array[k-i];
    }
    if(psa_width){
        Interval T(0.0, psa_width);
        Interval z = 0.0;
        int s=a.num;
        int s1 = a.num + b.num - 2;
        for(int i=s1; i>=s; i--){
            Interval sum=0.0;
            int jmin = __max(0, i-b.num+1);
            int jmax = __min(i, a.num-1);
            for(int j=jmin; j<=jmax; j++)
                sum += a.array[j] * b.array[i-j];
            z = z * T + sum;
        }
        ret.array[s-1] += z * T;
    }
    return ret;
}

psa operator * (const double& a, const psa& b)
{
    return psa(b.num, a, ARG_TERM)*b;
}

psa operator * (const psa& a, const double& b)
{
```

```
    return a*psa(a.num, b, ARG_TERM);
}

psa operator / (const psa& a, const psa& b)
{
    return a*Inv(b);
}

psa operator / (const double& a, const psa& b)
{
    return psa(b.num, a, ARG_TERM)/b;
}

psa operator / (const psa& a, const double& b)
{
    return a/psa(a.num, b, ARG_TERM);
}

psa operator ^ (const psa& a, const int& b)
{
    return Pow(a, b);
}

psa Pow(psa x, int y)
{
    int i;
    Interval a=x.array[0];
    psa a2(x.num, ARG_TERM);
    a2.array[0]=a;
    psa h(1, 0.0);
    h = x - a2;

    psa r(x.num, ARG_TERM);
    r.array[0] = Pow(a, y);
    psa hn(x.num, 1.0, ARG_TERM);
    double fact_n = 1.0;
    double coeff = 1;
    int index = y;
    Interval a_index = r.array[0];
    for(i=1; i<x.num-1; i++){
        hn = hn * h;
        fact_n /= i;
        coeff *= index;
        index -= 1;
        a_index = a_index / a;
        psa temp(x.num, ARG_TERM);
```

```
        temp.array[0]=a_index;
        r = r + fact_n * coeff * temp * hn;
    }

    if(x.num == 1) return r;

    hn = hn * h;
    fact_n /= i;
    coeff *= index;
    index -= 1;
    a_index = a_index / a;
    if(psa_width){
        Interval range = x.range();
        psa temp(x.num, ARG_TERM);
        temp.array[0]=Pow(range, index);
        r = r + fact_n * coeff * temp * hn;
    }else{
        psa temp(x.num, ARG_TERM);
        temp.array[0]=a_index;
        r = r + fact_n * coeff * temp * hn;
    }
    return r;
}

psa Inv(psa x)
{
    Interval a = x.array[0];
    psa h(1, 0.0), a2(x.num, ARG_TERM);
    a2.array[0]=a;
    h= x - a2;

    psa r(x.num, ARG_TERM);
    r.array[0] = 1/a;
    psa hn(x.num, 1.0, ARG_TERM);
    double sign = 1;
    Interval xn = 1/a;
    for(int i=1; i<x.num-1; i++){
        hn =hn * h;
        xn = xn / a;
        sign *= -1;

        psa xn2(x.num, ARG_TERM);
        xn2.array[0] = xn;
        r = r + sign * xn2 * hn;
    }
    if(x.num == 1) return r;
```

```
    hn =hn * h;
    xn = xn / a;
    sign *= -1;
    if(psa_width){
        Interval psa_range = x.range();
        xn = 1/(Pow(psa_range, x.num));
    }
    psa xn2(x.num, ARG_TERM);
    xn2.array[0] = xn;
    r =r + sign * xn2 * hn;
    return r;
}

ostream& operator << (ostream& s, const psa& a)
{
    s << "[" ;
    for(int i=0; i<a.num; i++)
        s << a.array[i] << " ";
    s << "]";
    return  s;
}
```

```
/* ------------------------------------ <mat.h> ---- */
class matx;
class mat
{
public:
    int num;
    ppsa *array;

    mat();
    mat(int dim);
    mat(const mat& obj);
    virtual ~mat();
    void clean();
    void copy(const mat& cp);

    mat& operator = (const mat&);
    psa& operator [](int nIndex);

    friend mat operator + (mat, mat);
    friend mat operator - (mat, mat);
    friend mat operator - (mat);
    friend mat operator * (matx, mat);
    friend ostream& operator << (ostream&, const mat&);
};

/* ------------------------------------ <mat.cpp> ---- */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <assert.h>
#include "seclib.h"
#include "psa.h"
#include "mat.h"

mat::mat()
{
    cout << "mat: Don't use default!" << endl;
    exit(0);
}

mat::mat(int dim)
{
    num=dim;
    array=0;
    if(dim!=0){
        array=(psa**)malloc(sizeof(psa*)*dim);
        for(int i=0; i<dim; i++)
```

```
            array[i]=new psa(1, ARG_TERM);
    }
}

mat::mat(const mat& cp)
{
    copy(cp);
}

mat::~mat()
{
    clean();
}

void mat::clean()
{
    if(array!=0){
        for(int i=0; i<num; i++)
            delete array[i];
        free(array);
    }
}

void mat::copy(const mat& cp)
{
    num=cp.num;
    if(cp.array!=0){
        array=(psa**)malloc(sizeof(psa*)*num);
        for(int i=0; i<num; i++){
            array[i]=new psa(1, ARG_TERM);
            *(array[i])=*(cp.array[i]);
        }
    }else array=0;
}

mat& mat::operator = (const mat& cp)
{
    clean();
    copy(cp);
    return *this;
}

psa& mat::operator [](int nIndex)
{
    return *(array[nIndex]);
}
```

```
mat operator + (mat a, mat b)
{
    assert(a.num==b.num);
    mat ret(a.num);
    for(int i=0; i<a.num; i++){
        int dim=__max(a[i].num, b[i].num)-1;
        ret[i]=a[i].setorder(dim)+b[i].setorder(dim);
    }
    return ret;
}

mat operator - (mat a, mat b)
{
    assert(a.num==b.num);
    mat ret(a.num);
    for(int i=0; i<a.num; i++){
        int dim=__max(a[i].num, b[i].num)-1;
        ret[i]=a[i].setorder(dim)-b[i].setorder(dim);
    }
    return ret;
}

mat operator - (mat a)
{
    mat ret(a.num);
    for(int i=0; i<a.num; i++)
        ret[i]=(-1)*a[i];
    return ret;
}

ostream& operator << (ostream& s, const mat& a)
{
    for(int i=0; i<a.num; i++)
        s << *(a.array[i]) << endl;
    return s;
}
```

```
/* ------------------------------------ <matx.h> ---- */
class matx
{
public:
    int num;
    mat **array;

    matx();
    matx(int dim);
    matx(int y, int x);
    matx(const matx& obj);
    virtual ~matx();
    void clean();
    void copy(const matx& cp);

    matx& operator = (const matx&);
    mat& operator [](int nIndex);

    friend matx operator * (matx, matx);
    friend mat operator * (matx, mat);
    friend matx operator - (matx, matx);
    friend matx operator - (matx);

    friend matx inv(matx&);
    friend matx tenchi(matx&);
    friend ostream& operator << (ostream&, const matx&);
};
```

```
/* ------------------------------------ <matx.cpp> ---- */
#include <stdlib.h>
#include <math.h>
#include <assert.h>
#include "seclib.h"
#include "psa.h"
#include "mat.h"
#include "matx.h"

matx::matx()
{
    cout << "matx: Don't use default!" << endl;
    exit(0);
}

matx::matx(int dim)
{
    num=dim;
    array=0;
```

```
    if(dim!=0){
        array=(mat**)malloc(sizeof(mat*)*dim);
        for(int i=0; i<dim; i++)
            array[i]=new mat(dim);
    }
}

matx::matx(int y, int x)
{
    num=y;
    array=0;

    if(num!=0){
        array=(mat**)malloc(sizeof(mat*)*num);
        for(int i=0; i<num; i++)
            array[i]=new mat(x);
    }
}

matx::matx(const matx& cp)
{
    copy(cp);
}

matx::~matx()
{
    clean();
}

void matx::clean()
{
    if(array!=0){
        for(int i=0; i<num; i++)
            delete array[i];
        free(array);
    }
}

void matx::copy(const matx& cp)
{
    num=cp.num;
    if(cp.array!=0){
        array=(mat**)malloc(sizeof(mat*)*num);
        for(int i=0; i<num; i++){
            array[i]=new mat(0);
```

```
            *(array[i])=*(cp.array[i]);
        }
    }else array=0;
}

matx& matx::operator = (const matx& cp)
{
    clean();
    copy(cp);
    return *this;
}

mat& matx::operator [](int nIndex)
{
    return *(array[nIndex]);
}

matx operator - (matx a, matx b)
{
    assert(a.num==b.num);
    matx ret(a.num);
    for(int i=0; i<a.num; i++)
        ret[i]=a[i]-b[i];
    return ret;
}

matx operator - (matx a)
{
    matx ret(a.num);

    for(int i=0; i<a.num; i++)
        for(int j=0; j<a.num; j++)
            ret[i][j]=(-1)*a[i][j];
    return ret;
}

matx operator * (matx a, matx b)
{
    assert(a.num==b.num);
    matx ret(a.num);

    for(int i=0; i<a.num; i++){
        for(int n=0; n<a.num; n++){
            psa temp(1, 0.0);
            for(int j=0; j<a.num; j++){
                int dim=__max(a[i][j].num, b[j][n].num)-1;
```

```
                temp=temp.setorder(dim)+a[i][j].setorder(dim)*b[j][n].setorder(dim);
            }
            ret[i][n]=temp;
        }
    }
    return ret;
}

mat operator * (matx a, mat b)
{
    assert(a.num==b.num);
    mat ret(b.num);

    for(int i=0; i<a.num; i++){
        psa temp(1, 0.0);
        for(int j=0; j<a.num; j++){
            int dim=__max(a[i][j].num, b[j].num)-1;
            temp=temp.setorder(dim)
                +a[i][j].setorder(dim)*b[j].setorder(dim);
        }
        ret[i]=temp;
    }
    return ret;
}

/*２× ２の行列を転置する関数*/
matx tenchi(matx& x)
{
    matx r(2);
    r[0][0]=x[0][0];
    r[0][1]=x[1][0];
    r[1][0]=x[0][1];
    r[1][1]=x[1][1];
    return r;
}

/*２× ２の行列の逆行列を求める関数*/
matx inv(matx& x)
{
    matx r(2);

    if(x.num!=2){
        cout << "入力ミス" << endl;
        return (-1);
    }
```

```
    psa delta(1, 0.0), invdel(1, 0.0);
    delta=x[0][0]*x[1][1]-x[0][1]*x[1][0];
    invdel=1/delta;
    r[0][0]=invdel*x[1][1];
    r[0][1]=(-1)*invdel*x[0][1];
    r[1][0]=(-1)*invdel*x[1][0];
    r[1][1]=invdel*x[0][0];
    return r;
}

ostream& operator << (ostream& s, const matx& a)
{
    for(int i=0; i<a.num; i++)
        s << *(a.array[i]);
    return s;
}
```