max-min方程式の初期値問題を解くCプログラムの実装

大島, 良太郎
早稲田大学基幹理工学研究科

高橋, 大輔
早稲田大学基幹理工学研究科

https://doi.org/10.15017/2924861

出版情報：応用力学研究所研究集会報告. 2019AO-S2, pp.102-107, 2020-03. 九州大学応用力学研究所
バージョン：
権利関係：

応用力学研究所研究集会報告 No.2019AO-S2
「非線形波動研究の多様性」（研究代表者　永井　敦）

Reports of RIAM Symposium No.2019AO-S2

Diversity in the research of nonlinear waves

Proceedings of a symposium held at Chikushi Campus, Kyushu University,
Kasuga, Fukuoka, Japan, October 31 - November 2, 2019

Article No. 18 (pp. 102 - 107)

max-min方程式の初期値問題を解くCプログラムの実装

大島 良太郎（Oshima Ryotarou），高橋 大輔（TAKAHASHI daisuke）

Research Institute for Applied Mechanics
Kyushu University
March, 2020

max-min 方程式の初期値問題を解く C プログラムの実装

早稲田大学基幹理工学研究科　大島良太郎 (OSHIMA Ryotarou)
早稲田大学基幹理工学研究科　高橋大輔 (TAKAHASHI Daisuke)

概要

max, min, 符号反転 − のみで構成される任意の表現を，ある標準形に同値変形するアルゴリズムを提案し，C プログラムによって実装する．また，max-min 方程式の初期値問題についてそのプログラムを応用し，計算効率を議論する．

1　はじめに

与えられた式表現を一定の形式に帰着させることは数学のさまざまな場面で用いられる重要な操作であり，そのためのアルゴリズムは公式の導出や記号処理ソフトの自動計算で非常に役に立つ．簡単な例を挙げると，有理関数を通分して約分したり，あるいは，部分分数展開することは，式表現の一定の形式への還元であり，そのためのアルゴリズムは重宝する．2 次曲線は座標変換により放物線，双曲線，楕円に帰着する．また，可積分系分野とその周辺では，ソリトン方程式は双線形形式に変換することで KP ヒエラルキーに組み込むことができ，動く特異点を持たない非線形常微分方程式として Painlevé 方程式という 6 つの標準形がある．

本稿では，max 演算に関する一定範囲の式表現をある標準形に帰着させるためのアルゴリズムを提案し，それを計算機上で実行するための C プログラムの開発について述べる．max 演算を含む式表現は全順序束や max-plus 代数で主役となるものであり，通常の四則演算の公式と対応する公式も多く存在するが，お互いに使えたり使えなったりする公式も多い．したがって，max 演算に特化した式変形のアルゴリズムやプログラムを開発することは，max を含む式の解析には重要であり，本稿でも時間発展方程式の解の予想への応用を報告する．

2　max 表現の標準形への変形アルゴリズム

$\mathbb{R}$ 上の max, min および符号反転 − の演算と実変数で構成される表現（max 表現と呼ぶ）を考える．たとえば u_1, u_2, ... を実変数として

$$\min(u_1, -\min(-u_2, u_3, \max(u_4, -\max(u_5, u_6), \max(-u_1, u_4)))) \tag{1}$$

である．このような表現は，以下で定義される「標準形」に必ず帰着させることができる．

【標準形】

$$\min(a_1, a_2, \ldots, \max(A_1), \max(A_2), \ldots)$$

あるいは min と max が入れ替わった

$$\max(a_1, a_2, \ldots, \min(A_1), \min(A_2), \ldots)$$

この標準形の詳細は以下の通りである．

- min (max) の中に max (min) が 0 個以上含まれており，min と max の入れ子はそれ以上多重には存在しない．
- a_i は任意の変数 (たとえば x)，あるいはその符号反転（たとえば $-x$）であり，A_i はそれらの 2 個以上の並びである．
- 任意の i, j に対して $a_i \neq a_j$, $a_i \notin A_j$，かつ，$A_i \not\subset A_j$．
- 特殊な場合として，演算の入れ子がない $\min(a_1, a_2, \ldots)$, $\max(a_1, a_2, \ldots)$ や，単項 a_1 という場合もある．

任意の表現を上の標準形に変形する方法はいくつか考えられが，本稿では以下の 1, 2, 3, 4 のステップを順に行う方法を採用する．a_i は任意の項 (min, max, 変数，それらの符号反転)，A_i は 0 個以上の項の並びとする．

1. 次式の左辺から右辺への変形を再帰的に行う．

$$-\min(a_1, a_2, \ldots) = \max(-a_1, -a_2, \ldots), \qquad -\max(a_1, a_2, \ldots) = \min(-a_1, -a_2, \ldots)$$

   このステップが終了すると，$-\min$, $-\max$ という表現部分がなくなる．

2. 次式の左辺から右辺への変形を再帰的に行う．

$$\begin{aligned}\min(A_1, \min(a_1, a_2, \ldots), A_2) &= \min(A_1, a_1, a_2, \ldots, A_2),\\ \max(A_1, \max(a_1, a_2, \ldots), A_2) &= \max(A_1, a_1, a_2, \ldots, A_2)\end{aligned}$$

   このステップが終了すると，min, max の入れ子は min と max が交互に入れ子になっているものだけとなる．

3. 次式の左辺から右辺への変形を行い，必要ならステップ 2 を行う．この操作を再帰的に行う．

$$\begin{aligned}&\min(A_1, \max(A_2, \min(a_1, a_2, \ldots), A_3), A_4)\\ &\quad = \min(A_1, \max(A_2, a_1, A_3), \max(A_2, a_2, A_3), \ldots, A_4),\\ &\max(A_1, \min(A_2, \max(a_1, a_2, \ldots), A_3), A_4)\\ &\quad = \max(A_1, \min(A_2, a_1, A_3), \min(A_2, a_2, A_3), \ldots, A_4)\end{aligned}$$

   このステップが終了すると，min と max の入れ子はせいぜい一重となり，

$$\begin{aligned}&\min(a_1, a_2, \ldots, \max(A_1), \max(A_2), \ldots)\\ &\text{もしくは}\\ &\max(a_1, a_2, \ldots, \min(A_1), \min(A_2), \ldots)\end{aligned}$$

   という形を得る．

4. ステップ 3 で得られた表現に対して，以下の操作を再帰的に行う．
   - $a_i \in A_j$ となる i, j の組があれば $\max(A_j)$ ($\min(A_j)$) を取り除く．
   - $A_i \subset A_j$ となる i, j の組があれば $\max(A_j)$ ($\min(A_j)$) を取り除く

また，次の操作は上の各ステップで適宜行う．

- $a_1, a_2, \ldots$ に同じ項があればひとつを残して他を取り除く．
- 同様に A_i に含まれる項に同じものがあればひとつを残して他を取り除く．

- A_i に含まれる項がひとつだけであれば，$\ldots, \max(A_i), \ldots$ を $\ldots, A_i, \ldots$ ($\ldots, \min(A_i), \ldots$ を $\ldots, A_i, \ldots$) とする．

以上の操作を任意の表現に対して順に行うと標準形が得られる．なお，上の各ステップで用いられる変形は min, max の公式を用いた同値変形であり，元の表現と標準形の表現は等価であることを断っておく．

(1) についてこれらステップを行った結果を以下に示す．（項の並びは適宜変えている．）

$$\begin{aligned}
&\min(u_1, -\min(-u_2, u_3, \max(u_4, -\max(u_5, u_6), \max(-u_1, u_4))))\\
&\quad\downarrow \quad \text{ステップ } 1\\
&\min(u_1, \max(u_2, -u_3, \min(-u_4, \max(u_5, u_6), \min(u_1, -u_4))))\\
&\quad\downarrow \quad \text{ステップ } 2\\
&\min(u_1, \max(u_2, -u_3, \min(u_1, -u_4, \max(u_5, u_6))))\\
&\quad\downarrow \quad \text{ステップ } 3\\
&\min(u_1, \max(u_1, u_2, -u_3), \max(u_2, -u_3, -u_4), \max(u_2, -u_3, u_5, u_6))\\
&\quad\downarrow \quad \text{ステップ } 4\\
&\min(u_1, \max(u_2, -u_3, -u_4), \max(u_2, -u_3, u_5, u_6))
\end{aligned}$$

3　C プログラムのデータ構造

以上のアルゴリズムを C プログラムによって実装するには，max 表現のデータ構造が必要となる．我々は max 表現を以下のポインタを含む構造体として実現した．

```
typedef struct _list{
int mono_or_op;  // +1 なら変数，-1 なら min，max
int sign;        // +1 なら符号が +，-1 なら符号が-
int site;        // 整数値なら変数を区別する添え字，NULL なら min，max
int op;          // +1 なら max，-1 なら min，NULL なら変数
struct _list *arg;     // min，max の引数へのポインタ
struct _list *parent;  // 自分を引数として持つ親演算を指すポインタ
struct _list *prev;    // 自分が属する引数の列のひとつ前の項を指すポインタ
struct _list *next;    // 自分が属する引数の列のひとつ後の項を指すポインタ
} list;
```

例として表現 $\min(u_3, \max(-u_0, u_{-2}, u_0))$ を考えると，図 1 のデータ構造となる．矢印はポインタを表している．

4　アルゴリズムによるデータ構造の変化

次に，第 2 節で述べたアルゴリズムを用いてデータ構造がどのように変化するかについて簡単に触れる．たとえば次式の左辺から右辺への変形は第 2 節のアルゴリズムのステップ 3 によって実行される．

$$\min(\max(\min(A, B), C), D) = \min(\max(A, C), \max(B, C), D)$$

プログラムによってデータ構造がどのように変化していくかを図 2 に示す．

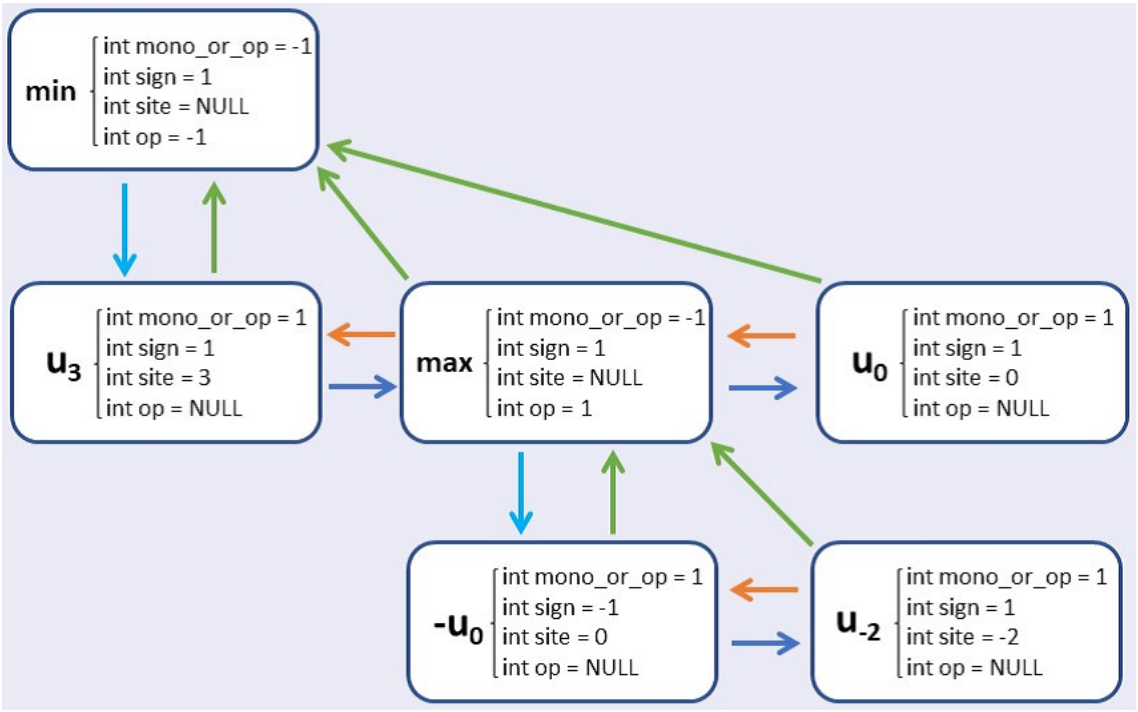
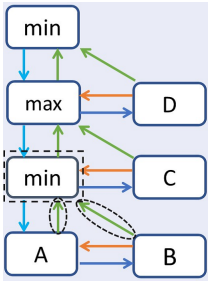
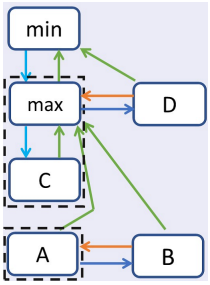


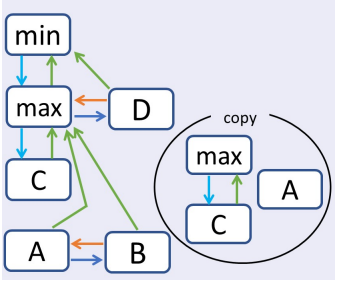
図 1　$\min(u_3, \max(-u_0, u_{-2}, u_0))$ のデータ構造



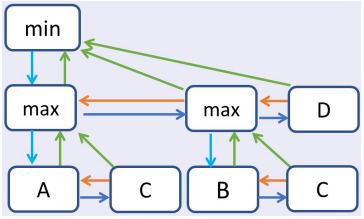
1. min を削除しポインタをつなぎかえる



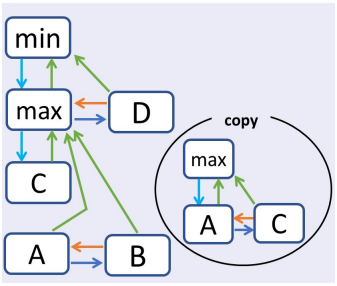
2. max, C, A をコピー



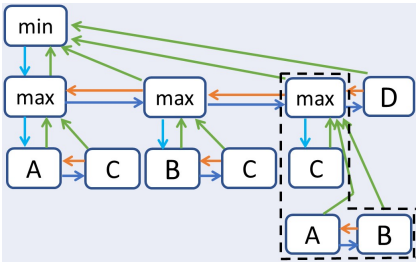
3. コピーを再接続



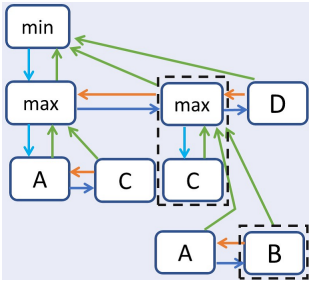
4. できた構造を本体に接続



5. max, C, B について同じ操作を繰り返す



6. 元データの削除



7. 終了

図 2　$\min(\max(\min(A,B),C),D)$ から $\min(\max(A,C),\max(B,C),D)$ への変形

5　プログラムによる応用例

以下の max-min 方程式を考える．

$$u_j^{n+1} = \min(\max(-u_{j-1}^n, u_j^n), u_{j+1}^n) \tag{2}$$

時刻 $n=0$ を初期時刻とすると，u_j^1 は方程式をそのまま用いて $\{u_i^0\}$ によって

$$u_j^1 = \min(\max(-u_{j-1}^0, u_j^0), u_{j+1}^0)$$

と表せる．次に，u_j^2 は方程式を再帰的に適用して

$$\begin{aligned} u_j^2 &= \min(\max(-u_{j-1}^1, u_j^1), u_{j+1}^1) \\ &= \min(\max(-\min(\max(-u_{j-2}^0, u_{j-1}^0), u_j^0), \min(\max(-u_{j-1}^0, u_j^0), u_{j+1}^0)), \min(\max(-u_j^0, u_{j+1}^0), u_{j+2}^0)) \end{aligned}$$

となる．以降の n でも方程式を再帰的に適用すると，u_j^n は $O(3^n)$ のオーダーの $\{u_i^0\}$ の項数によって形式解が得られる．（解の表現において，同じ u_i^0 であっても，現れる位置が異なるものは別々に数えるとする．）ところが先述のアルゴリズムによって項の整理を行うと，たとえば u_j^2 は

$$u_j^2 = \min(u_2^0, \max(-u_0^0, u_1^0), \max(-u_{-1}^0, -u_0^0, u_0^0))$$

と項数が 9 から 6 に減る．さらに u_j^3 は

$$u_j^3 = \min(u_3^0, \max(-u_1^0, u_2^0), \max(-u_0^0, -u_1^0, u_1^0), \max(-u_{-1}^0, -u_0^0, -u_1^0, u_0^0))$$

と整理でき，形式解の項数 27 が 10 に減る．任意の n での解は文献 [1] に報告されており，以下で与えられる．

$$u_j^n = \min(\max(-u_{j-1}^0, -u_j^0, \ldots, -u_{j+n-3}^0, -u_{j+n-2}^0, u_j^0), \max(-u_j^0, -u_{j+1}^0, \ldots, -u_{j+n-2}^0, u_{j+1}^0), \\ \ldots, \max(-u_{j+n-3}^0, -u_{j+n-2}^0, u_{j+n-2}^0), \max(-u_{j+n-2}^0, u_{j+n-1}^0), u_{j+n}^0) \tag{3}$$

この解の項数は n について多項式オーダー $O(n^2)$ であり，形式解の指数オーダー $O(3^n)$ から激減する．(2) の右辺は max, min, 符号反転 $-$ のみで構成されており，本稿の対象とする表現である．u_j^n を多項式オーダーの初期値によって表せる場合は，解の振る舞いをクリアに示すことができ，項の整理をしない形式解と比べて解についての情報が格段に増す．ところが，このような形式で記述される時間発展方程式について，u_j^n を与える一般的な解法はまだ見つかっていない．u_j^1, u_j^2, u_j^3, ... について，上のように具体的に $\{u_i^0\}$ で表し，項を整理して一般の n での形を予想し，帰納法などによって解の証明を行うというのが，現実的に考えられる有効な手立てとなる．

この際に，小さい n での具体形を整理して表示できるコンピュータプログラムは，手計算の限界をカバーする強力なツールとなる．さらに，Mathematica などの記号処理ソフトでも同様の計算が行えるが，用いる演算や公式が限られる今回のような記号処理においては，C プログラムなどの低レベルの言語で機能を特化した方が量的にも速度的にも圧倒的に有利となる．以下に本稿で用いた C プログラムと同様の処理を行う Mathematica プログラムの実行時間の比較を示す．表の数値は計算時間を秒で示したものである．

	C	Mathematica
u_j^3	0.0001	0.0060
u_j^4	0.0002	0.0227
u_j^5	0.0005	0.0747
u_j^6	0.0006	0.4389

	C	Mathematica
u_j^7	0.0011	3.1453
u_j^8	0.0014	24.9921
u_j^9	0.0019	247.2540
u_j^{10}	0.0026	2274.5335

	C	Mathematica
u_j^{20}	0.0140	計測不能
u_j^{30}	0.0473	計測不能
u_j^{40}	0.0914	計測不能
u_j^{50}	0.1830	計測不能

以上の例は max 表現を標準形に還元することの意義を示しており，また，C プログラムで実装することの有効性も確認された．しかしながら，再び (2) 方程式の例に戻ると，解 (3) は

$$u_j^n = \min(u_{j+n}^0, \max(-u_{j+n-2}^0, \min(u_{j+n-1}^0, \max(-u_{j+n-3}^0, \min(u_{j+n-2}^0, \max(-u_{j+n-4}^0, \ldots))))))$$

という形式でも表すことが可能で，項数は $O(n^1)$ となって (3) よりもオーダーの次数がひとつ下がる．これは想定している標準形とは異なる形式であり，時間発展方程式に限らず max 表現において何らかの意味を持つ形式が多く存在することの傍証となっている．通常の四則演算で表される多項式にさまざまな形式が存在するように，max 表現を自由自在に操ることのできる汎用ツールを作れないかというのが今後の重要な目標である．

参考文献

[1] T.Ikegami, D.Takahashi, J.Matsukidaira, “On solutions to evolution equations defined by lattice operators”, Japan J. Indust. Appl. Math. 21 (1013) 111-120, 1013