情報処理学会論文誌 Vol.56 No.5 1417–1427 (May 2015)

正規表現を用いた数式検索手法の提案

渡部 孝幸[1,a)] 宮崎 佳典[2]

受付日 2014年9月3日, 採録日 2015年2月4日

概要：数式は多くの科学分野において広く用いられているものであり，電子的な文書中に記述されることも多い．電子的な文書の特長として，検索を行うことができるという点があげられる．しかし，数式の検索を行うことは容易ではない．なぜなら数式では，通常の言語とは異なり，文字が二次元的に配置されるためである．そこで本研究では，二次元的な文字の構造に対して文字列のパターンマッチング（文字列探索）に基づく検索を行うことで，文書から特定の数式が記述された箇所を見つけ出す，文書内検索の機能を実現する手法を提案する．また，本研究で提案する手法は，数式のパターンマッチングにおいて正規表現を利用することも可能である．正規表現を用いることで，複雑なパターンを処理することが可能となり，検索の利便性が飛躍的に向上する．さらに，パターンにマッチした数式のハイライト表示および数式の置換の実装についても述べる．

キーワード：数式検索，正規表現，MathML

Pattern Matching Algorithm for Mathematical Expressions with a Regular Expression

Takayuki Watabe[1,a)] Yoshinori Miyazaki[2]

Received: September 3, 2014, Accepted: February 4, 2015

Abstract: Mathematical expressions are commonly described in scientific documents. Electronic documents have made it simple to retrieve its contents. However, retrieving mathematical expressions correctly is still challenging because characters in mathematical expressions are two-dimensionally located with their structures. In this study, we propose a pattern matching algorithm for mathematical expressions, which is similar to string searching algorithms. This method enables users to find mathematical expressions that they input in a large document. It also allows them to use a regular expressions for complex and flexible patterns of mathematical expressions. Additionally we have implemented the functions to highlight and replace them, based on the proposed method.

Keywords: mathematical expression retrieval, regular expression, MathML

1. はじめに

数式は，数学的な概念やモデルを記述するために，多くの科学分野において広く用いられている．電子的な文書においても数式が記述されており，web ページや電子教科書などで数式を目にする機会は多い．電子的な文書の特長として，検索が可能であるという点をあげることができる．数式は，日本語や英語と同じく，文字を配置することで情報を表現しているため，通常の言語に類する検索を行うことができると考えられる．しかし，通常の言語とまったく同一の方法で検索を行うことはできない．これは，数式では文字が二次元的に配置されるためである．

数式の検索に関しては，多くの研究が行われている．それらの研究は，複数の数式の中からユーザが望んでいるであろう数式を探し出して提示する，ということを目的としており，数式の間の類似性や，数式の持つ数学的な特徴を

[1] 静岡大学創造科学技術大学院・日本学術振興会特別研究員 DC
Graduate School of Science and Technology, Shizuoka University/Research Fellow of Japan Society for the Promotion of Science, Hamamatsu, Shizuoka 432–8011, Japan
[2] 静岡大学大学院情報学研究科
Graduate School of Informatics, Shizuoka University, Hamamatsu, Shizuoka 432–8011, Japan
[a)] dgs13012@s.inf.shizuoka.ac.jp

考慮することを重視している [1], [2], [3], [4], [5].

一方，数式の検索を行う際，「1 つの文書中で，ある数式が記述されている位置を探す」ということを目的とする場合がある．このような検索は，通常の言語に対する検索における文書内検索に類するものである．文書内検索を用いると，数百ページの書籍や web ページから，目的とする情報が記述されている位置を瞬時に見つけ出すことができる．文書内検索には，通常，パターンマッチング（文字列探索）が用いられる．

また，通常の言語におけるパターンマッチングでは，正規表現の機能が備えられていることも多い．正規表現を用いると，柔軟にパターンを記述することができるようになるため，正規表現を用いて数式を検索することができれば，高い利便性をもたらすと考えられる．そこで本研究では，数式に対して，正規表現を用いてパターンマッチングを行う手法を提案する．これに類する研究として，数式検索にワイルドカードの機能を導入することが試みられているが [6], [7]，文字列の正規表現とは機能，記法ともに大きく異なるものとなっており，文字列の正規表現ほどパターンの記述能力を向上させるものではない．本研究では，文字列の正規表現と同様の機能，記法の正規表現を，数式において実現する．なお，本研究で提案する数式の正規表現は，後方参照の機能も備えている．これにより，文書内検索において柔軟なパターンを記述できるだけでなく，数式を含む文書を作成している際に数式の置換を行うといった目的においても，本手法が高い利便性を発揮する．

以下，2 章において本研究で対象とする数式のためのデータ形式である MathML について述べる．3 章でパターンの作成方法とマッチング規則，およびハイライト表示と置換の機能を示す．4 章で提案手法の実装方法を詳述する．5 章で先行研究と比較しつつ本手法の特徴を示し，6 章で結語と今後の展望を述べる．

2. MathML

本研究でパターンマッチングの対象となる数式は，MathML [8] の Presentation Markup と呼ばれる形式で作成されているものとする．MathML は W3C によって策定された形式であり，広く普及している．HTML5 にも含まれていることから，特に web ページにおける標準的な形式であるということができる．文書内検索は web ページにおいて利用される機会が多いため，本研究が対象とする形式として MathML は適切なものである．また，LaTeX などの他形式から MathML に変換するツールはすでに研究されているため（文献 [9] など），MathML 以外の形式で作成された数式に本システムを応用することも可能であると考えられる．MathML には，Content Markup と呼ばれる形式もある．Content Markup は，数式処理ソフトウェアにおけるデータの交換形式としての利用などを想定されており，文書中に記述されることはきわめて稀である．このため本研究では Content Markup は対象としない．以下，特に断らない限り，MathML とは MathML Presentation Markup を指す．

```
<math>
  <mfrac>
    <mn>1</mn>
    <mi>x</mi>
  </mfrac>
</math>
```

図 1 MathML データの例
Fig. 1 An example of MathML data.

MathML は XML ボキャブラリの一種であり，数式はタグを用いて記述される．MathML による $\frac{1}{x}$ の記述例を**図 1** に示す．mfrac の第 1 の子が分子，第 2 の子が分母となる．mn は数，mi は識別子を意味する．

3. 機能

提案手法は，パターンと数式とのマッチングを行うものである．マッチした部分のハイライト表示と置換を行う機能を持つ．

本研究で利用できる正規表現は，連接，閉包（量化），和（選言）という基本的な機能だけでなく，ワイルドカードや後方参照の機能も備えている．特に後方参照は，数式のパターンマッチングにおいて，パターンの表現能力を飛躍的に高める．後方参照を用いることによって，たとえば「分母が同一の分数の和」といったパターンを記述することが可能となる．

本手法ではパターンを文字列として記述する．数式における記号の位置構造のための記法は，LaTeX の記法に類したものとしている．正規表現の記法は，標準的な正規表現の記法と同一である．このため，計算機上で数式を記述した経験を持ち，正規表現の記法を知っているユーザであれば，本手法を容易に利用することができる．また，パターンが文字列であるため，パターンを自動生成して本手法に対する入力とするといったパイプライン処理が可能であり，本手法をモジュールとしてシステムに組み入れることができる．一方，パターンを文字列として入力するという方法は，煩雑さや入力の誤りの原因となりうる．この問題は，パターンを生成するような GUI をフロントエンドとして実装することで解決することができると考えられる．

以下，パターンを記述するための記法，およびマッチングの規則について述べ，パターンおよびハイライト表示された数式の例を示す．また，置換の機能についても述べる．

3.1 文字および構造

文字としては，unicode で定義された文字を使用することができる．そのまま文字を入力することもでき，文字実

表 1 構造の一覧
Table 1 A list of structures.

構造名 (keyword)	数式の例	contents 1	contents 2	contents 3	MathML の要素名
下付き文字 (sub)	$_1$	下付き文字			msub
上付き文字 (sup)	^{2}	上付き文字			msup
上下付き文字 (subsup)	$_0^\infty$	下付き文字	上付き文字		msubsup
下文字 (under)	$\lim_{n\to\infty}$	基底	上部の文字		munder
上文字 (over)	$\vec{a}$	基底	下部の文字		mover
上下文字 (underover)	$\sum_{i=0}^{n}$	基底	下部の文字	上部の文字	munderover
分数 (frac)	$\frac{1}{2}$	分子	分母		mfrac
平方根 (sqrt)	$\sqrt{x}$	根号の内部			msqrt
冪根 (root)	$\sqrt[3]{a}$	根号の内部	根指数		mroot

体参照を用いることもできる（e.g.，和集合の記号として，∪ と∪ という 2 種類の記述方法が可能であり，これらは同一の文字を表す）．unicode には数式に用いられる記号のためのカテゴリが用意されており，総和（Σ），総乗（Π），偏微分記号（∂），積分記号（$\int$，$\oint\!\!\!\!\int$）などはもちろん，幾何学や論理学，集合論などで用いられる記号を含むパターンも記述できる．

数式中の文字の構造は，以下の形式で記述する．なお，角括弧記号（[,]）は「存在しない場合もある」ことを意味する．

```
¥keyword{contents1}[{contents2}[{contents3}]]
```

keyword の種別によって，keyword の後に続く contents の数は決まっている．また，contents は構造を含むことができる．つまり，構造の入れ子が許される．たとえば，分数の場合，keyword は frac であり，contents を 2 つとる．平方根の keyword は sqrt であり，contents は 1 つである．$\frac{1}{\sqrt{2}}$ は，次のように表現される．

```
¥frac{1}{¥sqrt{2}}
```

keyword は 9 種類用意されている．**表 1** に，keyword，contents1，2，3 の文字の配置，数式の例，および keyword に対応する MathML の要素名を示す．

表 1 に示した要素は記号の位置を指定するための MathML3.0 の要素のうちの主要なものを含む．記号の位置を指定する要素として，表 1 にあげたもの以外では，文字の左側に添字を付す要素（mmultiscript），行列や記号の整列（上下に並んだ方程式の等号の位置揃えなど）に関する要素（mtable，mlabeledtr，mtr，mtd，maligngroup，malignmark），筆算を表現するための要素（mstack，mlongdiv，msgroup，msrow，mscarries，mscarry，msline）があげられる．本手法では現在，これらの構造をパターンとして入力することはできない．これらの構造に含まれる記号を検索するには，これらの構造に対応する MathML の要素の子に対して本手法を適用すればよい．このほかの MathML の要素のうち，mi，mn，mo，mfenced については後述する．空白を挿入するもの，MathML データのエラーに関するもの，文字に装飾を施すもの，動的な機能を付加するものなどについては，記号の構造とは直接的には関係しないため，本研究では対象とせず，検索対象の MathML データ中に含まれる場合は取り除く．なお，本研究で扱う要素は，MathML1.0 および MathML2.0 においても定義されているため，これらのバージョンにおいても本手法は適切に動作する．

マッチングにおいては，構造は 1 つの文字として扱われる．また，構造と通常の文字，keyword が異なる構造どうし，および keyword が同一で contents が異なる構造どうしは，異なる文字と見なされる．

マッチングは，数式全体だけでなく，構造の contents に対しても行われる．たとえば，パターンが $\frac{1}{2}$，数式が $\sqrt{a+\frac{1}{2}}$ の場合，sqrt の contents である $a+\frac{1}{2}$ に対してもマッチングを行うため，$\sqrt{a+\frac{1}{2}}$ のうちの $\frac{1}{2}$ の部分のみがマッチする．

3.2 正規表現

本システムで用いることができる正規表現の機能は，以下のとおりである．

- ワイルドカード（任意の 1 文字，任意の 1 構造，およびその両方）
- 量化
- 選言
- 文字クラス，否定文字クラス
- 後方参照

任意の 1 文字および任意の 1 構造は，メタ文字 '.' で表現される．また，任意の 1 文字は '¥c'，任意の 1 構造は '¥s' と表現される．

量化を用いると，直前に出現する表現の反復を指定できる．量化のために，3 つのメタ文字が用意されている．

- `*` 直前の表現が 0 個以上
- `+` 直前の表現が 1 個以上
- `?` 直前の表現が 0 個もしくは 1 個

選言は，複数の表現のうちのいずれかにマッチするようなパターンを作成するためのものである．メタ文字 '|' を用いて記述し，'|' の前あるいは後のパターンにマッチする．

量化および選言のスコープは，丸括弧記号‘(’，‘)’を用いて指定できる．たとえば，(abc)+ は，abc，abcabc などにマッチし，a(b|c)d は，abd あるいは acd にマッチする．

構造と量化および選言を併用する際，構造は単一の文字と見なされる．また，構造の contents において量化や選言を用いることもできる．つまり，たとえば ¥sqrt{.+}+ のようなパターンを作成できる．このパターンは，$\sqrt{a}\sqrt{b}$，$\sqrt{5a^2-2a+2}$ などにマッチする．

文字クラスおよび否定文字クラスは，「指定した文字のうちのいずれか 1 つ」にマッチするようなパターンを作成するための機能である．メタ文字‘[’，‘]’，‘^’を用いて表現する．文字クラスの場合，括弧‘[’，‘]’で囲まれた文字のうちのいずれかにマッチし，否定文字クラスの場合，括弧‘[^’，‘]’で囲まれた文字以外の文字のうちのいずれかにマッチする．ハイフン記号を用いた略記も可能であり，たとえば [0-9] は任意の数字，[^stx-z] は s，t，x，y，z 以外の文字を表す．文字クラスおよび否定文字クラスの括弧の中に構造および文字実体参照を含めることはできない．構造の contents の中に文字クラスおよび否定文字クラスを含めることはできる．

後方参照は，パターン中の一部（サブパターン）を指定し，サブパターンにマッチした数式と同一の数式を後から呼び出す機能である．パターンの指定は，スコープの指定と同じく，丸括弧記号‘(’，‘)’で囲むという方法で行われる．サブパターンは，開き丸括弧記号‘(’が最も左側のものを 1 として，順に番号付けされる．サブパターンにマッチした数式は，¥n で呼び出される．n には，サブパターンの番号を記述する．たとえば，¥frac{.+}{(.+)}-¥frac{.+}{¥1} というパターンには，分母が同一であるような分数の差，つまり $\frac{1}{2}-\frac{a+1}{2}$ などの数式がマッチする．

ここまでに述べたメタ文字のうち，.，*，+，?，|，(，)，[，]，{，}，^，&，;，¥ は，バックスラッシュ（本論文では円記号（¥）で示す）を用いてエスケープすることで，通常の文字と見なされる．なお，文字クラスおよび否定文字クラスの内部においては，前述の文字のうちのほとんどはメタ文字と見なされないため，エスケープなしに記述してよい．文字クラスおよび否定文字クラスにおいてエスケープする必要のあるメタ文字は [，]，^，-，¥ である．

3.3 パターンとマッチング結果の例

パターンと数式とのマッチング結果の例を**表 2** に示す．左列にパターン，右列にマッチした文字がハイライト表示された数式を示す．

3.4 置換

置換は，パターンにマッチした部分を，置換後の数式として指定した数式に置き換える機能である．

置換後の数式の記法を説明する．置換後の数式において，文字実体参照を用いた記号，構造，後方参照の記述を許す．このため，&，;，¥，{，} はメタ文字であり，これらを記号として入力する場合にはエスケープしなければならない．一方，置換後の数式は一意に定まる必要があるため，ワイルドカードや量化といった機能は使用できない．このため，置換後の数式では，.，*，+，?，_，(，)，[，]，^ はエスケープしなくてよい．

置換後の数式における後方参照は，パターン中のサブパターンを参照する．たとえば，パターンを ¥sqrt{(.+)}，置換後の数式を ¥1¥sup{¥frac{1}{2}} として，数式 $\sqrt{x}$ を置換した場合，結果は $x^{\frac{1}{2}}$ となる．

表 2 パターンとマッチング結果の例
Table 2 Examples of patterns and highlighted mathematical expressions.

aとbの乗算	
a[×·]?b	$a \times b$, $a \cdot b$, ab
微分	
(¥frac{d.*}{d.+}|¥sup{'}|¥over{.+}{'})	$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = v(t)$, $\left(af(x)\right)' = af'(x)$, $\ddot{x}(t) + \dot{x}(t) + t$
相加平均，相乗平均の関係	
¥frac{(.+)¥+(.+)}{2}≥¥sqrt{¥1[×·]?¥2}	$\frac{a+b}{2} \geq \sqrt{ab}$, $\frac{x_1+x_2}{2} \geq \sqrt{x_1 x_2}$
e^xのマクローリン級数	
e¥sup{x}=1¥+x¥+(¥frac{x¥sup{([0-9]+)}}{¥2!}¥+)+…	$e^x = 1 + x + \frac{x^2}{2!} + \cdots$ $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \cdots$ （なお，このパターンは $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^3}{3!} + \cdots$ といった数式にもマッチする）

3.5 想定される利用方法

ハイライト表示および置換が実際に利用される場面について述べる．

ハイライト表示は，web ページの閲覧の際に効果を発揮する．ハイライト表示を用いることで，目的とする記述が web ページ中に記載されている箇所を一目で把握することができる．主要な web ブラウザはすべて自然言語のためのハイライト表示機能を有していることからも，ハイライト表示が web ページ閲覧において必要とされていることが分かる．また，web ブラウザのハイライト表示はパターンマッチングに基づいている．本手法もパターンマッチングに基づいているため，既存のハイライト表示と同様の使用感で数式に対するハイライト表示を行うことができる．web ページ閲覧におけるハイライト表示の実用化には，web

ブラウザの拡張機能として本手法を実装するという方法が考えられる．

置換は，文書編集において有用である．たとえば，文書中に記述した数式中の記号 x を，すべて θ に書き換える必要が生じた際，置換を用いればすぐに書き換えを終えることができる．本手法では，置換後の文字列中で後方参照を使用することが可能であるため，3.4 節で述べたような，平方根による表記を $\frac{1}{2}$ 乗による表記に置換するといった高度な処理もできる．

4. 実装

本章ではまず，数式に対する正規表現を実現するうえでの問題点とその対処方法を述べる．その後，数式を木として表現する方法，および数式データを文字列に内部的に変換する方法を示す．また，前章で示した記法で記述されたパターンを，実際にマッチングを行う正規表現エンジンである Onigmo [10] に入力するパターンへと内部的に変換する方法についても述べる．

4.1 数式の正規表現における実装の問題点と対処方法

数式の記号の位置構造は，木構造として表現することができる．木構造は，括弧を用いて，文字列として一意に表現することができる．ここで，数式のパターンマッチングを行うための方法として，文字列化された木に対して正規表現を用いたマッチングを行うというものが考えられる．しかし，この方法では数式のパターンマッチングとして適切に動作しない．これは，正規表現に括弧の対応関係を扱う能力がないためである（なお，ここでいう括弧とは，構造の contents を囲む括弧のことであり，$(a+b)^n$ といった，数式における括弧のことではない．数式における括弧は，単なる 1 つの文字として扱われる）．この事実は形式言語理論において確かめられている．例をあげると，「任意の文字を含む平方根」を意図して作成された `¥sqrt{.+}` というパターンは，`¥sqrt{x}¥sqrt{y}` にマッチしてしまう．これは，パターン中の「任意の文字の 1 度以上の反復」を意味する `.+` が，文字列化された数式中の `x}¥sqrt{y` にマッチしてしまうためである．なお，これを避けるために，パターン中の `.+` を「`{`, `}` を除く任意の文字の 1 度以上の反復」を意味する `[^{}]+` に書き換えると，パターンが `¥sqrt{x¥sup{2}}` にマッチしなくなる．

正規表現を拡張して括弧の対応関係を取り扱うための方法の 1 つとして，木オートマトンを用いる方法がある [11]．しかし，木オートマトンを用いたマッチングには，後方参照の機能を導入することが容易ではない．

別の方法として，パターンを再帰的に利用できるよう拡張が施された正規表現を利用するという方法がある．このような正規表現を処理できる正規表現エンジンとして，Onigmo が知られている．Onigmo では後方参照を利用することも可能である．このため，本研究では，システムに入力された数式検索用のパターンを Onigmo で処理可能なパターンに変換し，かつ，数式データを文字列に変換したうえで，パターンと文字列を Onigmo に入力し，マッチング処理は Onigmo が行うという方法をとる．

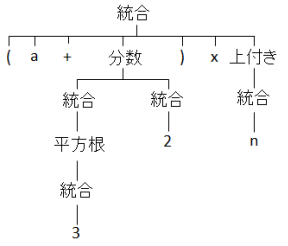

図 2 $\left(a+\frac{\sqrt{3}}{2}\right)x^n$ の数式木

Fig. 2 A mathematical expression tree for $\left(a+\frac{\sqrt{3}}{2}\right)x^n$.

4.2 数式から文字列への変換

数式の記号の位置構造は，葉を 1 つの文字（文字節），葉以外の節を，位置構造を指定するための節（構造節）および数式中の所定の位置に記述される文字列を統合するための節（統合節）とする木構造（数式木）として表現することができる．構造節は 9 種類あり，これらは表 1 に示した 9 種類の構造に対応する．構造節の子は必ず統合節である．統合節は，構造の contents に対応する．統合節の子は必ず構造節か文字節であり，子の数は任意である．例として，$\left(a+\frac{\sqrt{3}}{2}\right)x^n$ を数式木として表現したものを**図 2** に示す．図 2 において，構造節は分数，平方根，および上付きである．また，それぞれの統合節は，その位置に記述される文字列を子に持つ．この方法を用いることで，本研究で扱う 9 種類の記号の位置構造を含む数式を，一意に木に変換することができる．

なお，数式の木構造として，演算の優先順位を表現したものが知られているが，本研究における木構造は数式中の文字の位置構造を表現したものであり，これは演算の優先順位を示す木構造とはまったく異なるものである（なお，MathML Content Markup のデータから得られる木は，演算の優先順位を表現した木である）．

(1) MathML から数式木への変換（MathML データの整形）

MathML は XML ボキャブラリの 1 つであるため，MathML データを一意に木に変換することができる．しかし，MathML データの木は，数式木と共通する点は多いものの，異なる点がいくつかある．また，MathML データ（およびそれから得られる木）は，1 つの数式に対して一意に定まらない．実際に，使用する数式エディタが異なると，同一の数式を作成しても MathML データに違いが生じる．さらに，同じエディタを用いて同一の数式を記述したとしても，数式の入力の手順が異なると得られる MathML デー

```
<math>
  <mfenced>
    <mi>a</mi>
    <mo>+</mo>
    <mfrac>
      <mrow>
        <msqrt>
          <mn>3</mn>
        </msqrt>
      </mrow>
      <mn>2</mn>
    </mfrac>
  </mfenced>
  <msup>
    <mi>x</mi>
    <mi>n</mi>
  </msup>
</math>
```

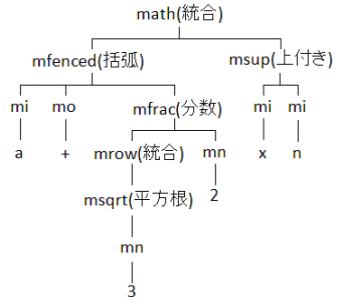

図 3 $\left(a+\frac{\sqrt{3}}{2}\right)x^n$ の MathML データとその木

Fig. 3 A MathML code for $\left(a+\frac{\sqrt{3}}{2}\right)x^n$ and its tree.

タが同一とならない場合がある．1 つの数式に対して異なる木が得られてしまうと，マッチング処理が困難になる．このため，MathML データに前処理を施し，数式木と同一の形に変換する．

ここで，MathML データから得られる木について，数式木と比較して述べる．$\left(a+\frac{\sqrt{3}}{2}\right)x^n$ を MathML で表現したもののうちの一例と，それに対応する木を，**図 3** に示す．

MathML において，構造節は表 1 に示した要素として定義されている．統合節は，mrow と呼ばれる要素に対応する（根の統合節は math 要素）．文字節は，mi，mn，mo という 3 種類の要素の子に対応する．mi，mn，mo はそれぞれ，子となる文字が識別子，数字，演算子のいずれであるかを区別するための要素である．

MathML が数式木と異なる点を，6 つ述べる．これらは，MathML から得られる木が一意に定まらない原因でもある．また，MathML と数式木を同一視するための処理についても述べる．

- 統合節（math, mrow）の子が統合節となることを許す．
- 構造節の子が統合節にならない（構造節あるいは文字節を直接子に持つ）ことを許す．

MathML のこれらの特徴によって，統合節が任意の位置に出現しうるため，木が一意に定まらなくなる．これを解決するために，構造節の直下が統合節でない場合は統合節を挿入し，構造節の直下以外の統合節はすべて削除する．

- 文字の種別を指定する

前述のように，MathML では，文字が識別子であるか，数字であるか，演算子であるかを，mi，mn，mo という 3 種類の要素で区別する．しかし，これらの要素の違いは数式の描画には反映されないため，構造を持つ文字列として数式を扱う本研究においては，これらの区別は不要である．このため，mn および mo は mi に変換する．

- 文字の「まとまり」を表現する

mi，mn，mo は，2 つ以上の文字を子に持つことが可能であり，これによって，文字のまとまりを表現する．例として，`<mi>sin</mi>` のような記述ができる．しかし，この文字列のまとまりという情報は，本研究では考慮しない．なぜなら，文字列のまとまりを考慮すると，たとえば abcd という数式が `<mi>ab</mi><mi>cd</mi>` という MathML データで表現されていた際，bc というパターンをこの abcd における bc にマッチさせられなくなるためである．この挙動は，混乱を招く原因となる．このため，複数の文字を子に持つ mi，mn，mo は分解する．つまり，`<mi>sin</mi>` は `<mi>s</mi><mi>i</mi><mi>n</mi>` となる．

- 上付き文字，下付き文字，上下付き文字（msup，msub，msubsup）が基底を持つ

MathML では，上付き文字，下付き文字および上下付き文字は，基底となる文字列と，上付き（下付き，上下付き）文字を指定するという形で表現される．しかし，上付き（下付き，上下付き）文字の基底は一意に定まらない場合がある．たとえば，ab^2 という数式を記述する際，上付き文字 2 の基底を，ab とすることも，b とすることもできる．ここで，パターンが b^2，数式が ab を基底とした ab^2 である場合に，これらをマッチさせられないことは望ましくない．このため数式木では，上付き（下付き，上下付き）文字は基底を持たないものとする．MathML データとしては，基底に相当する部分を msup (msub，msubsup) の兄要素に配置し，msup (msub，msubsup) の第 1 の子は，子を持たない mrow とする．

- 括弧を構造節として表現することができる

MathML では，mfenced 要素を用いて括弧を記述できる．しかし，本研究においては，括弧は構造と見なさず，括弧記号を単なる 1 つの文字と見なす．このため，mfenced を `<mi>(</mi>` および `<mi>)</mi>` に変換する．

MathML データを整形することで，上述のような数式木との違いをなくし，MathML データを数式木と同一のものとすることができる．文献 [12] では，Mathematica®，Microsoft Word®，LaTeXMathML [13] の 3 つの方法で得られた MathML データの違いを検討したうえで，正規化によってデータが一意となることを実験により確認している．

図 3 の MathML データおよび MathML の木に整形処理を施したものを**図 4** に示す．図 4 の整形後の MathML の木は，図 2 の数式木と同一の形になっていることが分かる．

(2) 数式木から文字列への変換

整形された MathML データから得られた一意な木を，正規表現エンジンで処理するために文字列（数式木文字列）に変換する．統合節は，その子を並べた文字列に変換される．統合節の子のうち，文字節は 1 つの文字に変換され，構造節は以下の文字列に変換される．

```
<math>
  <mi>(</mi>
  <mi>a</mi>
  <mi>+</mi>
  <mfrac>
    <mrow>
      <msqrt>
        <mrow>
          <mi>3</mi>
        </mrow>
      </msqrt>
    </mrow>
    <mrow>
      <mi>2</mi>
    </mrow>
  </mfrac>
  <mi>)</mi>
  <mi>x</mi>
  <msup>
    <mrow></mrow>
    <mrow>
      <mi>n</mi>
    </mrow>
  </msup>
</math>
```

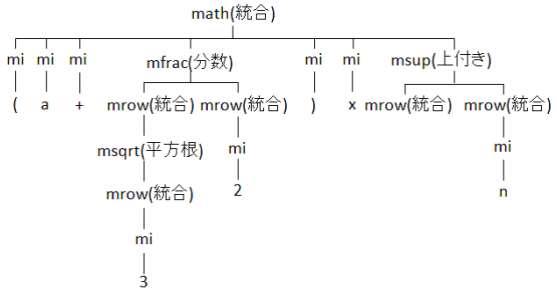

図 4 整形された MathML データとその木
Fig. 4 A formatted MathML data and its tree.

表 3 構造の別名
Table 3 Aliases of structures.

構造節名	別名	構造節名	別名
underover	:::::::::	frac	::::
subsup	::::::::	sqrt	:::
under	:::::::	sub	::
over	::::::	sup	:
root	:::::		

/構造節名/{統合節 1/}[/{統合節 2/}[/{統合節 3/}]]

{統合節 2} および {統合節 3} が記述されるのは，それが存在する場合のみである．統合節が構造節を子に持つ場合は，括弧（{，}）が入れ子となる．例として，$\left(a+\frac{\sqrt{3}}{2}\right)x^n$（図 2 の元となる数式）を文字列に変換すると，次のようになる．

(a+/frac/{/sqrt/{3/}/}/{2/})x/sup/{n/}

この文字列をそのまま正規表現エンジンに入力すると，パターンが構造節名や括弧記号にマッチしてしまう可能性がある（たとえばパターンが「a」のとき，構造節名 frac の a にマッチする）．このため，構造節名を別名に置き換える．別名の一覧を**表 3** に示す．

$\left(a+\frac{\sqrt{3}}{2}\right)x^n$ から得られる文字列に対して別名への置き換えを行ったものを以下に示す．

(a+/::::/{/:::/{3/}/}/{2/})x/:/{n/}

また，文字節としての {，}，:，¥ は，それぞれ，¥{，¥}，¥:，¥¥ に置換する．ここで，統合節の内容を囲む括弧の直前にスラッシュ（/）が挿入されている理由を述べる．スラッシュを挿入しないと，たとえば数式が √¥ である場合，これを変換して得られる文字列/:::{¥¥} が ¥} という文字列を含んでしまうため，パターンが「}」の場合，構造節を表現するための括弧にマッチしてしまう．スラッシュを挿入しておくことで，これを避けられる．

4.3 ユーザが作成したパターンから Onigmo へ入力するパターンへの変換

ユーザが作成したパターン（ユーザパターン）を，Onigmo に入力するパターン（Onigmo パターン）に変換する方法を述べる．パターンの変換は，4.2 節 (2) で述べた数式から文字列への変換に対応するための変換，括弧の対応を扱うための変換，後方参照のための変換の 3 つに大別できる．それぞれについて述べる．

(1) 数式の文字列化に対応するための変換

ユーザパターン中の構造に対しては，前節で述べた，数式木の構造節の変換と同様の変換を行う．すなわち，¥keyword{contents1}[{contents2}[{contents3}]] は，以下のように変換される．なお，alias は表 3 に示した別名である．

/alias/{contents1/}[/{contents2/}[/{contents3/}]]

さらに，置換後の構造はすべて丸括弧で囲む．これは，量化子のスコープを適切に設定するためである．ユーザパターンにおいては，利便性のため，たとえば `¥sqrt{2}+` という記法を許しているが，このパターンにおける量化子 '+' は，Onigmo パターンにおいては「} が 1 回以上連続する」ことを意味するものとして扱われる．構造全体を括弧で囲むことで，量化子が「構造全体が 1 回以上連続する」ことを意味するものと解釈される．

構造を表現するための文字である，:，{，}，/ の処理について述べる．ユーザパターンが文字 : を含む場合，これを `¥¥:` に置換する．また，`¥{`，`¥}` は，`¥¥{`，`¥¥}` に変換する．ユーザパターンが文字 / を含む場合，これを `/(?![{}:])` に置換する．この Onigmo パターンは，正規表現の否定的先読みと呼ばれる機能を用いており，「直後の文字が，{，}，: のいずれでもないような /」を表現している．この変換によって，文字として記述された / は，構造を表現するための文字の接頭辞としての / にはマッチしなくなる．

(2) 括弧の対応関係を扱うための変換

まず，括弧の対応関係を扱うための変換に用いる Onigmo の機能を説明する．本研究で用いる機能は，「名前付き捕獲式集合」および「名前指定呼び出し」と呼ばれるものである．名前付き捕獲式集合とは，サブパターンに名前を割り当てる機能であり，名前指定呼び出しとは，名前を割り当てたサブパターンを呼び出す機能である．サブパターンを再帰的に呼び出すことも可能であり，これによって括弧

の対応関係を扱うことができる．

そもそも，¥(.+¥) が (a)(b) のような文字列にマッチしてしまうのは，任意の 1 文字を表す ‘.’ が括弧記号にマッチしてしまうためであった（なお，‘.’ を [^()] とすると上述の問題は回避できるが，括弧が入れ子になった場合に外側の括弧にマッチしなくなる）．このため，任意の 1 文字または 1 構造を意味する ‘.’，任意の 1 文字を表す ‘¥c’，任意の 1 構造を表す ‘¥s’ を，上述の機能を用いた Onigmo パターンに変換する必要があるということになる．

まず，‘.’ について述べる．‘.’ は，次のパターンに変換される．可読性のためにインデントをつけて示す．

```
(
  (?<ac>[^{}:/¥¥]|¥¥{|¥¥}|¥¥:|¥¥¥¥|/(?![{}:]))
  |
  (?<as>/:+(/{(¥g<ac>|¥g<as>)+/})+)
)
```

(?<name>p) という記法は，パターン p に名前 name を割り当てることを意味する．

ac（arbitrary character）と名付けられたパターンは，

- 構造に関わる 5 つの文字（{，}，:，/，¥）以外の文字
- ¥{
- ¥}
- ¥:
- ¥¥
- 直後が{，}，: でない /

のいずれかにマッチする．構造に関わる 5 つの文字が，単なる文字として数式中に出現する場合，数式木を文字列に変換する処理において，直前にバックスラッシュが付与された形に変換されている．このため，このパターンは，数式における任意の 1 文字にマッチする．

as（arbitrary structure）と名付けられたパターンについて説明する．:+ は，keyword の別名にマッチする．contents に対応する部分を見ると，¥g<name> という記法は，name と名付けられたパターンを呼び出すためのものである．つまり，/{(¥g<ac>|¥g<as>)+/} は，「(contents を意味する）括弧の内部は，1 つ以上の任意の文字または任意の構造」であることを意味する．このパターンにおいて，パターン as の内部で as を再帰的に呼び出している．これにより，括弧の対応関係を扱うことができる．さらに，contents に相当する部分を括弧で囲み，量化子 ‘+’ を付しているため，contents が複数存在する場合にも as は構造に適切にマッチする．

メタ文字 ‘.’ は，単なる文字にも構造にもマッチするため，‘.’ は，ac と as のいずれか，という形で表現される．

任意の文字にマッチする ‘¥c’ は，上記のパターンのうちの ac と名付けられた部分に変換され，任意の構造にマッチする ‘¥s’ は，パターン as の部分に変換される．なお，1 度パターンを作成した後は，名前指定呼び出しでパターンを呼び出せばよいため，たとえば同一のパターン中に，‘.’ と ‘¥c’ がともに記述された場合，‘¥c’ は ¥g<ac> に変換される．

(3) 後方参照のための変換

Onigmo パターンにおける後方参照の記法を説明する．サブパターンの指定には，前述の (?<name>p) の記法を用いる．サブパターンにマッチした部分の呼び出しには，名前指定参照と呼ばれる機能を用いる．この機能は ¥k<name> と記述される．ここで，前述の ¥g<name> はサブパターンそのものを呼び出すが，¥k<name> はサブパターンにマッチした部分を呼び出す．たとえば，(?<s>.)¥g<s>，および (?<s>.)¥k<s> という 2 つのパターンと文字列 ab があるとき，前者は ab にマッチするが，後者はマッチしない．なぜなら，前者のパターンにおける ¥g<s> は ‘.’（つまり任意の文字）として扱われるが，後者のパターンにおける ¥k<s> は a として扱われるためである．

ユーザパターンから Onigmo パターンへの変換においては，ユーザパターン中の丸括弧記号のすべてに，名前を割り当てる．つまり，(*pattern*) というパターンを，(?<br*n*>*pattern*) に変換する．なお，*n* は数字であり，最も左側に開き括弧がある括弧を 1 として，右に向かって順に数字を割り当てる．ユーザパターンにおける，サブパターンにマッチした部分の呼び出し ¥*n* は，¥k<br*n*> に置き換える．

パターンの変換処理において多くの括弧が追加されるが，ユーザパターンに記述されていた括弧には特別な名前を割り当て，サブパターンにマッチした部分は名前指定参照で呼び出すという方法により，変換処理で追加された括弧を無視することができる．

4.4 マッチング結果を用いたハイライト表示と置換

ここまでに述べた方法で，数式木文字列と Onigmo パターンが得られる．数式木文字列に対して Onigmo パターンをマッチングするという処理を行うと，数式木文字列中のどの箇所に Onigmo パターンが記述されたかを特定することができる．本節では，マッチング結果を用いてハイライト表示および置換を行う方法を述べる．

(1) ハイライト表示

ハイライト表示には，MathML の要素 mstyle を用いる．mstyle は，数式中の記号の背景色やフォントカラーなどを指定するための要素である．MathML データ中の mstyle の子孫にあたる部分に，mstyle で指定した装飾が施される．今回は，マッチした箇所の背景色を黄色にするという形でハイライト表示を行うため，mathbackground 属性の値を yellow に指定した mstyle を用いる．

ハイライト表示を行うためには，ハイライト表示したい部分の親として mstyle を挿入すればよい．これはすなわち，数式木において構造節を新たに導入し，ハイライトさ

れるべき要素をその構造節の子とするということである．以後，ハイライトのために導入した構造節を highlight と呼ぶ．

パターンにマッチした箇所を highlight の子とするために，正規表現エンジンの置換機能を用いる．まず，Onigmo パターン全体を，(?<name>p) の記法を用いてサブパターンとして指定する．name は br0 とする．すなわち，Onigmo パターンを p とすると，新たなパターンは (?<br0>p) となる．置換後の文字列は /::::::::::/{¥k<br0>/} とする．置換後の文字列に名前指定参照を用いているため，¥k<br0> は Onigmo パターンにマッチした部分となる．このようにして得られたパターンと置換後の文字列を用いて置換処理を行うと，Onigmo パターンにマッチしたすべての部分が /::::::::::/{，/} で囲まれる．なお，:::::::::: は highlight の別名とする．これによって，パターンにマッチした部分を構造節 highlight の子とすることができる．

この置換の結果として得られた文字列に対して，4.2 節 (2) で述べた数式木（正規化された MathML データ）から数式木文字列への変換の，逆方向の変換を行う．数式と数式木文字列とは 1 対 1 に対応するため，逆変換は可能である．逆変換を行う際，highlight は mathbackground 属性の値を yellow に指定した mstyle に変換する．この処理によって，マッチした部分がハイライトされた数式（mstyle が挿入された MathML データ）を得ることができる．

(2) 置換

置換を行うために，まず，ユーザが入力した置換後の数式を変換する．構造節およびメタ文字は，4.3 節 (1) で述べた方法と同様の方法で変換する．なお，構造節全体を括弧記号 (，) で囲む処理，および / を /(?![{}:]) に変換する処理は行わない．後方参照は，4.3 節 (3) と同様の方法で変換する．

このようにして得られた置換後の文字列と Onigmo パターンを Onigmo に入力し，置換処理を行う．置換後の文字列に対して，ハイライト表示の場合と同様，4.2 節 (2) で述べた変換の逆変換を行う．逆変換によって，置換された数式を得ることができる．

5. 議論

ここまでに述べた，正規表現を用いた数式のマッチングの特徴を，先行研究と比較しつつ述べる．

5.1 数式中のパターンの位置の特定

本手法の特長として，数式中のパターンの位置を特定できるという点があげられる．たとえば，x というパターンを入力したら，マッチングされる数式中のどの位置に x が存在するかを特定できる．これによって，ハイライト表示や，置換といった応用を実現することが可能となる．

一方，数式の間の類似性に着目した検索などの場合，問合せの数式全体と，検索対象の数式全体との間の関係を見るため，検索結果は数式全体であり，数式中のどの位置に問合せを含むか，といった情報は得られない．

5.2 検索結果の順序付け

多くの先行研究で実現されている機能として，検索結果の順序付けがある．検索結果を順序付けすることで，大量の数式から曖昧な記憶に基づいて数式を探し出したり，ユーザにとって有用な数式の発見を促したりすることが可能となる．一方，本研究が目的とする文書内検索では，検索結果として提示すべきものは（数式中の）文字であり，数式そのものではない．すなわち，数式を順序付けするという機能がそもそも必要ではないため，本手法においてはこの機能を実装していない．

検索結果の順序付けを行う研究として，文献 [1] では，MathML Content Markup データに基づく木構造から得られる根から葉へのパスの集合に着目し，集合間の Jaccard 係数を数式の類似性と見なして検索結果を順序付けている．文献 [2] では，MathML Content Markup で記述された数式に対して数式の部分の形や関数の引数といった条件を利用した検索を行い，検索結果を，式の大きさ，一致部分の大きさ，一致部分の大きさの割合，tf-idf を利用して順序付けている．文献 [3] では，式の大きさと，MathML Content Markup データに基づく木の親子関係に着目した順序付けを行っている．文献 [4] では，MathML Content Markup データの要素の数を要素とするベクトルを構築し，ベクトル間のコサイン類似度を数式の類似度ととらえて数式を順序付けている．

5.3 数式における文字の数学的な性質の考慮

数式の検索を行ううえで，数学的な性質が考慮される場合は多い．それらの処理のうち，複数の研究で扱われているものをあげる．

(1) 識別子の同一視

数式においては，識別子の記号の違いは特別な意味を持たない．たとえば，$\sin x$ と $\sin\theta$ は，多くの場合，同一であると見なされる．この性質を考慮し，問合せに含まれる x は，x や θ ともマッチさせるという処理を行うような検索システムもある．このような研究として，文献 [2], [3], [5] があげられる．

本研究においてこのようなマッチングを行いたい場合は，文字クラスを用いるという方法がある．[a-zA-Zα-ωA-Ω] と入力すれば，任意の英字およびギリシャ文字にマッチさせることができる．添字が付された文字も識別子であると見なしたい場合は，[a-zA-Zα-ωA-Ω]¥sub{.+}|[a-zA-Zα-ωA-Ω] とすればよい．

(2) 文字の意味の考慮

数式において，文字が特別な意味を持つ場合がある．たとえば，＋や∩は演算子であり，a や x は識別子を意味する場合が多い．また，$\sin x$ における s は sin という関数の名称の一部であり，$s+t$ における s は 1 つの識別子である．

本手法では，文字の意味は考慮しない．これは，意味を考慮するためには推測が必要となるためである．たとえば sinhx という数式があった場合，三角関数 $\sin hx$ なのか，双曲線関数 $\sinh x$ なのか，あるいは s，i，n，h，x という 5 つの識別子の積なのか，数式の表記からは判別できない．ここで，検索システムによる意味の推測がユーザの判断と異なっている場合，ユーザが検索結果として得たい数式が得られないという事態が生じる．パターンマッチングに基づく検索において，この問題は利便性を大きく低下させると考えられる．なお，演算子と識別子を区別するようなパターンは，文字クラスを用いて擬似的に表現できる．たとえば，「2 つの識別子の間の四則演算」を意図した問合せは，`([a-zA-Zα-ωA-Ω]¥sub{.+}|[a-zA-Zα-ωA-Ω])[+-÷×・]?([a-zA-Zα-ωA-Ω]¥sub{.+}|[a-zA-Zα-ωA-Ω])` などとすればよい．

(3) 数学的な性質を考慮した検索と置換

数学的な性質を考慮した検索手法を用いて置換を実装することは不可能ではないが，置換の利便性が大きく低下すると考えられる．5.3 節 (1) の識別子の同一視を行うと，x という問合せに対して a や θ もマッチさせてしまうため，x だけを θ に置換するといった処理ができない．5.3 節 (2) の文字の意味の考慮を行うと，sinhx という数式が実際は $\sin hx$ を意図して記述されたものであるにもかかわらず $\sinh x$ であると推測されてしまった場合，sin が sinh にマッチしないため，sin を cos に置換するといった処理を正しく行うことができない．

5.4 XML データのためのツール

XML データから特定の要素を探し出し処理するための技術として，XSLT [14] と XQuery [15] があげられる．MathML は XML の一種であるため，これらの技術を MathML に対して利用することも可能である．しかしこれらは，XML 全般に対して汎用的に利用でき，多様な機能を持つため，利用方法の習得が容易ではない．また，これらの技術の利用方法を知っていたとしても，MathML は 1 つの数式に対してデータが一意に定まらないという性質を持つため，数式検索のための正しい問合せを記述するためには，4.2 節 (1) で述べた MathML の詳細を把握している必要がある．このため，MathML の検索の手段としてこれらを利用することは現実的ではない．

本手法は，数式のパターンマッチングに特化しているため，問合せの作成は比較的容易である．また，本手法では MathML データの正規化を行うため，ユーザは MathML データについて意識する必要はない．

6. 結語

本研究では，正規表現を用いて数式のパターンマッチングを行う手法を提案した．従来の数式検索は数式全体を検索結果として提示していたが，本手法はパターンにマッチした数式中の文字の位置を提示することができる．提案した手法を用いて，マッチした部分のハイライト表示および数式の置換を行う機能を実装した．

数式の文字の位置構造は木構造として表現されるため，通常の正規表現では取り扱うことができない．本研究では，パターンを再帰的に呼び出せるよう拡張された正規表現エンジン Onigmo を用いることで，数式に対する正規表現を実現した．

今後の課題として，パターンの入力を補助する GUI の開発があげられる．本手法では，パターンを文字列として入力するが，この方法は入力の誤りの原因となる．パターンの作成のための GUI を用意することで，パターンを正しく作成することが容易になると考えられる．

謝辞 本研究は，日本学術振興会特別研究員奨励費 26-2758 の助成を受けたものである．

参考文献

[1] Yokoi, K. and Aizawa, A.: An Approach to Similarity Search for Mathematical Expressions Using MathML, *2nd Workshop Towards Digital Mathematics Library*, pp.27–35 (2009).
[2] 小田切健一，村田剛志：MathML を用いた数式検索，第 22 回人工知能学会全国大会，1F1-3 (2008).
[3] 高田真澄，村尾裕一：数式の構造を反映した検索法，第 2 回データ工学と情報マネジメントに関するフォーラム，C7-4 (2010).
[4] 中西崇文，岸本貞弥，村方 衛，大塚 透，櫻井鉄也，北川高嗣：数式データを対象とした複合連想検索システムの実現，日本データベース学会 Letters, Vol.4, No.1, pp.29–32 (2005).
[5] Miner, R. and Munavalli, R.: An Approach to Mathematical Search Through Query Formulation and Data Normalization, *Towards Mechanized Mathematical Assistants*, pp.342–355, Springer (2007).
[6] Miller, B.R. and Youssef, A.: Technical Aspects of the Digital Library of Mathematical Functions, *Annals of Mathematics and Artificial Intelligence*, Vol.38, No.1-3, pp.121–136, Springer (2003).
[7] Altamimi, M.E. and Youssef, A.S.: Wildcards in Math Search, *Implementation Issues*, CAINE/ISCA, pp.90–96 (2007).
[8] MathML, available from ⟨http://www.w3.org/TR/MathML3/⟩.
[9] TtM, a TeX to MathML translator, available from ⟨http://hutchinson.belmont.ma.us/tth/mml/⟩.
[10] Onigmo, available from ⟨https://github.com/k-takata/Onigmo/⟩.
[11] Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S. and Tommasi, M.: Tree Automata Techniques and Applications (1997), available

from ⟨http://www.grappa.univ-lille3.fr/tata⟩.
[12] 渡部孝幸，宮崎佳典：二次元の位置構造に着目した数式のパターンマッチング手法，情報知識学会誌，Vol.22, No.3, pp.253–271 (2012).
[13] LaTeXMathML, available from ⟨http://math.etsu.edu/LaTeXMathML/⟩.
[14] XSLT, available from ⟨http://www.w3.org/TR/xslt20/⟩.
[15] XQuery, available from ⟨http://www.w3.org/TR/xquery/⟩.

渡部 孝幸 （学生会員）

2013 年静岡大学大学院情報学研究科修士課程修了．同年同大学創造科学技術大学院入学，現在に至る．数式コンテンツ処理，数学教育の研究に従事．日本学術振興会特別研究員 DC．日本応用数理学会学生会員．

宮崎 佳典 （正会員）

静岡大学大学院情報学研究科准教授，静岡大学創造科学技術大学院准教授（兼担）．1999 年 7〜8 月米国 UCLA 客員研究員．2013 年 4〜9 月米国ノースカロライナ州立大学客員研究員．博士（工学）．日本応用数理学会，日本 e-Learning 学会，教育システム情報学会，外国語教育メディア学会，情報知識学会，日本ソーシャルデータサイエンス学会各会員として，e-Learning，英語&数学教育ツール開発，数値解析周辺の研究に従事．