記　号　処　理　76－5
（１９９４．　１１．　１８）

公式データベースによる不定積分の求解パッケージ

高橋 岳之　長野 英二　斎藤 制海

豊橋技術科学大学

数式処理研究においては、解析的手法 (アルゴリズムによる解法) が主流となっており、各分野で大きな成果をあげている。しかし、特殊関数や複雑な式の中には、現在のアルゴリズムでは不定積分が求められないものも多く存在する。そこで、積分公式を用いて不定積分を解くためのシステムを開発したので報告する。

本システムは、数式処理システム GAL(*General Algebraic Language/Laboratory*) 上で作成し、積分公式データベースと公式運用モジュールで構成されている。

Package for solving indefinite integrals by using fomula database

Takeyuki Takahashi　Eiji Chono　Osami Saito

Toyohashi University of Technology

The analytical algorithms are the main approach to solving indefinite integral in the computer algebra system. However, they can not be applied to some of special functions and complex functions. In this study, the computer algebra system has been developed to obtain indefinite integrals of these functions by using the integral formula database.

This system is constructed on the computer algebra system GAL(*General Algebraic Language/Laboratory*) and consists of two parts, that is, integral formula database and formula operation module.

1 はじめに

数式処理システムとしては、現在 Reduce, Mathematica などが有名である。しかし、これらの数式処理システムでは代数計算をアルゴリズム (解析的手法) によって行なっているため、現在もなお効率的なアルゴリズムが発見されてない演算も存在し、例えば不定積分では、複雑な式や特異な形をした式などは現在のアルゴリズムでは解が得られないものも多い。ところが、人間はこのような不定積分も、基本的な公式や部分積分法、置換積分法等の解法のテクニックを用いることにより実際に解いている。現在、数学公式集という題の本がいくつか出版されているが、その中の不定積分の部分を見ると、いろいろな形の関数に対する不定積分が記述されている。この公式を数式処理システムで利用すれば、解析的手法で解けない不定積分の解を求められる可能性がでてくる。本研究では、不定積分の公式データベースと公式の運用モジュールを数式処理システム GAL[1](*General Algebraic Language/Laboratory*) 上で作成し、従来とは異なる手法を用いた不定積分求解システムを構築した。

なお、GAL は、広範囲の数学演算を実行することを目的とした数式処理システムであり、現在筑波大学等で開発が進められている。

2 積分公式データベース

積分公式データベースを構築する際に使用した公式は岩波書店の数学公式 I [5] に掲載されている公式であり、初等関数の不定積分の中から 764 個の公式をデータベースに登録した。

2.1 公式の特徴量

データベースのキーとなる公式の特徴量として以下のものを使用した。

公式の構造に関する特徴量

これは、数式が何で (どんな関数で) 構成されているかを示すものであり、キーワードとキーワードのレベルで表されている。

キーワード 数式のキーワードとは、数式中に含まれる擬関数名 (integ, sum 等)、純関数名 (sin, log等)、超越的数 (π等) とする。

キーワードのレベル 数式において、他のキーワード (主に擬関数名及び純関数名) の引数部以外の箇所に現れているキーワードを第 1 レベルのキーワードとする。キーワード K を第 F レベルのキーワードとするとき、K の引数部に第 1 レベルで出現しているキーワードを $F+1$ レベルのキーワードとする。

公式の性質に関する特徴量

単項 $v_1^{d_1}\cdots v_m^{d_m}f_1^{e_1}\cdots f_n^{e_n}$ $(d_i>0; e_j>0;\ i=1,\cdots,m; j=1,\cdots,n)$ において $v_1,\cdots,v_m$ を変数、$f_1,\cdots,f_n$ をキーワードとする。このとき、

$$\texttt{\#DEG}=\sum_{i=1}^{m}d_i+\sum_{j=1}^{n}e_j$$

キーワードの次数 $\deg(f_j)=e_j$

である。一般に、多項式 $T_1+\cdots+T_n$ (T_i : $i=1,\cdots,n$ は単項) において、#DEG $=\max\{\texttt{\#DEG}(T_i)|i=1,\cdots,n\}$ である。また、特殊な場合におけるカウント規則は以下のとおりである。ただし、F は自明でない (キーワードを含む) 数式とする。

(a) F^n (n は定数) : #DEG = #DEG(F) $\times n$ とする。キーワードについては F 内のキーワードの次数をそのままカウントする。

(b) $\sqrt{\mathrm{F}}$: #DEG = #DEG(F) $\times\frac{1}{2}$ とし、キーワードの次数は (a) と同様とする。

(c) $n\times$ M (M は多項式、n は数) :
#DEG = #DEG(M) とする。キーワードにつ

いては M 内のキーワードの次数をそのままカウントする。

(d) $\frac{N}{D}$: D が自明な数式の場合 (キーワードを含まない) には #DEG = #DEG(N) とし、D が自明でない場合は #DEG = #DEG(N) + #DEG(D) とする。いずれの場合にも、キーワードの次数については N 及び D 内のキーワードの次数をそのままカウントする。

(e) $X^1 + \cdots + X^n$ (n は不定) : 全次数が一意に与えられないので #DEG= "indef" とする。また、キーワードについても次数が定まらない場合には "indef" とする。

2.2 公式のインデックス

以下では、I 型、II 型のインデックスを定義する。

I 型のインデックス

I 型のインデックスは、公式の構造に関する特徴量を表すもので、積分公式の左辺 "integ(F,x)" に対して次のように作られる。

1. $f_1, \cdots, f_m$ を F 内の第 1 レベルのキーワードの集合とし、 $g_1, \cdots, g_n$ を F 内の第 2 レベルのキーワードの集合とする。以下、F 内にキーワードが出現しなくなるまでこの操作を行なう。
2. このとき、I 型のインデックスは次のように定義される。
   $((f_1 \ \cdots \ f_m) \ (g_1 \ \cdots \ g_n) \ \cdots)$

II 型のインデックス

II 型のインデックスは、公式の性質に関する特徴量を表すもので、積分公式の左辺 "integ(F,x)" に対して次のように作られる。

1. F 内の第 1 レベルのキーワードの集合を $k_1, \cdots, k_n$ とする。
2. F を全次数 (#DEG) 及び F 内のキーワードの次数によって以下のように特徴付ける。
   ((#DEG . 値) (k_1 . 値$_1$) $\cdots$ (k_n . 値$_n$))

2.3 検索方法

積分公式の検索は、逆ファイルを用いて行なう。逆ファイルとはインデックスに公式番号が対応するように作成されたファイルである。検索は、被積分関数から I 型、II 型のインデックスを生成し、索引である逆ファイル内を調べて公式番号を取り出すことにより行なわれる。

2 つのインデックスを用いた積分公式検索の手順を以下に示す。

step1 : 与えられた被積分関数に対して、I 型、II 型のインデックスを求める。

step2 : I 型のインデックスを使って、該当する公式番号の集合を取り出す。

step3 : II 型のインデックスをもとに、候補となる公式番号の集合を取り出す。

step4 : step2 、step3 で得られた集合の積集合をとる。

step5 : step4 で求めた番号の集合に対する公式を取り出す。

step6 : 公式 1 つ 1 つに対して被積分関数とのパターンマッチングを厳密に行ない、該当する公式を取り出す。

3 数式のパターンマッチング

3.1 積分公式の特徴

被積分関数と公式のパターンマッチングにおいては、以下のような特徴が見られる。

1. 公式は、積分変数を主変数とした初等関数の形に表されており、パターン変数 (定数 a,b 等) は係数とベキの部分にしか出現しない

2. パターン変数には定数のみがマッチングする

3. 積分変数以外の独立変数及びその関数は定数とみなすことができる

3.2 積分公式に対するパターンマッチング

3.1 節で述べた事柄から、積分公式に対するパターンマッチングの基本的な方法を考える。積分公式のパターンマッチングは

1. 積分変数を確定する

2. 積分変数に着目して、係数とベキ指数部のマッチングを行なう

という手順を基礎にして行なえばよい。また、係数とベキ指数部のマッチングにおいては以下のような基準を設けることにする。

1. 主変数のベキ指数が1以上である項の係数がパターン変数のとき、パターン変数の値は0でない。

2. 主変数のベキ指数がパターン変数のとき、パターン変数の値は0でない。

これは、公式を不必要に多く検索しないということと、パターンマッチャを簡素化するという理由で設けたものである。

3.3 パターンマッチングのアルゴリズム

積分公式のパターンマッチングは、最小単位式の集合を使って処理を行なう。方法としては、積分公式の中の最小単位式を1つずつ取り出し、それにマッチングする式が被積分関数の中に存在するかどうかを調べていくものである。その際に、パターン式(積分公式)のどの最小単位式から処理を行なうかという順序が問題になり、この順序によってマッチングの成否が決まる場合がある。そのため、マッチングを行なう順序をきちんと決定しておかなければならない。ここでは、どのような順序で処理を行なえばよいかについて述べる。

本システムにおいては、最小単位式の優先順位は次のように定義している。

$$\begin{aligned}&(\text{パターン変数を含まない式})\\ &\quad > (\text{パターン変数を含む式})\\ &\qquad > (\text{パターン変数の式})\end{aligned} \tag{1}$$

まず、マッチングの条件が厳しいパターン変数を含まない最小単位式の処理を最初に行なう。全体のマッチングが成功するときは、この最小単位式が必ず被積分関数の中に存在しなければならない。もし存在しなければ、マッチングは失敗となる。

次に行なうのはパターン変数を含む式の処理である。実際には、この中にもいろいろな順序付けがあるが、それについては後述する。

そして最後に、パターン変数の式の処理を行なう。これを1番最後にした理由は、パターン変数の式はどんな定数でもマッチングし、しかもマッチングする定数は必ずしも1個の最小単位式とは限らないからである。

さて、先ほどパターン変数を含む式の処理においても処理の順番が細かく決まっていると述べた。その順序は式(2)のようになっている。

$$\begin{aligned}&(\text{単項式})^{(\text{数値})} > (\text{単項式})^{(\text{変数})}\\ &> (\text{多項式})^{(\text{数値})} > (\text{多項式})^{(\text{変数})}\end{aligned} \tag{2}$$

ここで、関数は単項式と見なすことにする。これらはパターン変数を含む最小単位式であるため、$(\text{単項式})^{(\text{数値})}$ の "(単項式)" というのは引数部にパターン変数を含む関数であり、積分変数の単項式はパターン変数を含まない最小単位式に該当する(式の分解の規則による)。

多項式は、パターン変数の値のとり方によって単項式になるという曖昧さを持っている。そのため、順序を決めないで処理を行なうと多項式がマッチングしてはならない単項式とマッチングすることが起こりうる。そこで、曖昧さの少ない(条件の厳しい)単項式を先に処理してしまうことによって、多項式がマッチングしてはいけない単項式を事前に取り除き、多項式の処理において誤った対応でのマッチングを防ぐ。

(単項式)$^{(数値)}$ > (多項式)$^{(数値)}$ と (単項式)$^{(数値)}$ > (多項式)$^{(変数)}$ のマッチング順序についても同様で、単項式を優先する。

一般に、パターン変数が多く含まれている式ほどいろんな式とのマッチングが可能となるので、パターン変数をできるだけ含まない式から処理をする。

マッチングの手順

ここでは、マッチングの処理の実際の流れを示す。マッチング関数は mch(expr,ptn,prd) という形式で実行される。ここで、expr は被積分関数、ptn は公式パターン、prd はマッチングの条件を表した述語である。処理の手順は、次のようになる。

step1 : rexp と ptn の簡単なマッチングのチェックを行なう。これは、ptn 中のキーワードが rexp 内に存在するかを調べることにより行なう。

step2 : rexp と ptn を最小単位式に分解し、それぞれ R 、P という、最小単位式を要素とする集合にする。

step3 : P の要素のうち、パターン変数を含まない式を 1 つ 1 つ取り出し、その式が R 中に存在するかどうかを調べる。もし存在すれば、その式を R と P から取り除く。存在しなければマッチングは失敗となり終了する。

step4 : P から積分変数を含む式をすべて取り出し、P2 とする。パターン変数しか変数を含まない式は後回しにするので取り出さない。P2 を式 (2) の優先順位にしたがって並び変えた後、1 つ 1 つの式に対して R 中の式とマッチングを行なう。マッチングが成功すると R からマッチングした式を取り除く。R 中のどの式ともマッチングしなければ失敗となり終了する。

step5 : パターン変数式のマッチングを行なう。このとき P に残っている式の数は 0 か 1 個である。2 個以上ある場合は値が一意に決定できないのでマッチングを失敗させる。0 個のとき、 R に式が残っていればマッチングは失敗する。残ってなければマッチングは成功である。P に 1 個残っている場合は、R の式すべてとのマッチングを行なう。R が空ならば 1 とのマッチングを行なう。

step6 : マッチング結果を返して終了する。

step1 では、パターン式 (ptn) 中のキーワードに着目し、同じキーワードが被積分関数 (rexp) 内に存在するかを調べている。もしキーワードが存在しなければ全く関係のない式であるためマッチングは失敗する。例えば、被積分関数が対数関数の積分で、公式が三角関数の積分の場合、三角関数が被積分関数の中に含まれていないのでマッチングは失敗する。これは、明らかにマッチングしないとわかる式に対しては複雑なマッチング処理を行なわないようにするためのものである。

最小単位式が関数のときのマッチングは、関数名を比較した後、その引数に対して **step2** 以降の手続きを再帰呼出ししている。

4 積分公式データベース運用モジュールの構築

4.1 不定積分の求解

積分公式の選択・運用

データベースから公式を検索すると、複数個の公式が得られることがある。検索された公式は、いずれも与えられた問題に対して適用可能なものであるが、2 通りのケースがある。1 つは、どの公式を適用しても最終的な解が同一である場合である。このケースは、漸化式とその終了条件に関する公式が検索されたときに起こる。従って、この理由で複数の公式が検索されることは、公式の自動運用を目的としているシステムの性格上やむを得ないが、それ以外の場合は余分な公式をデータベースから取り除くべきかもしれない。

2つ目のケースは、適用する公式によって得られる解の表現が異なるというものである。このようなことは、ある式に対して等価な別の表現が存在するような三角関数等に起こる。漸化式の場合は、公式が適用できなくなるまで次数を減らした後、残った積分に対して再び公式を検索しなければならない。システムが返す答は1つでなければならないので、どれかを選択する必要がある。そこで、システムが人間の助けを借りずに自動で解を求めるために、公式を選択する基準を定めた。基準の内容については、公式の適用後の式の項数が少ないものを選ぶことにした。

このような評価は、公式が複数個検索されるたびごとに行なう。従って、漸化式の公式に対しては、完全な解が得られるまで公式を検索するのではなくて、その漸化式が適用できるまで適用し、残った積分はそのままにした状態で評価を行なう。

評価と選択の手順は、

1. 各公式の適用結果の式に対し、項1つにつき1点を加算する
2. 項の中に不定積分が含まれていれば、さらに10点を加算する。
3. 式を合計点の低い順番に並び変える。
4. 点数の一番低い式に対応する公式を選択する。

である。不定積分に対して10点を加算するのは、漸化式の公式をできる限り選ばないようにするためである。これは、データベース検索が時間のかかるものであることや、我々が公式を選択する場合でも漸化式よりは1度で解の求まる公式を採用するだろうという考えに基づくものである。なお、評価の点数が同じ場合には検索された順番、すなわちデータベースの前の方にある公式を選択している。

さて、ユーザによってはシステムが返す答に満足しないこともありうる。それは、ユーザにとって必要な数式というのが使用している状況によって変わることがあり、同じ評価基準で一律に判断することができないからである。そのような場合は、ユーザ自身が使いたい公式を選択することができる。これは、システムのフラグ (integauto) を "off" に切替えることにより可能となる。そのときは、毎回公式が検索されると、その公式と適用結果を表示し、実際に使用する公式をユーザに選択してもらうようになる。

不定積分の方法

本システムの不定積分求解の処理のながれは図1に示すようになる。公式を適用した後も不定積分が残るようなときは、その積分に対して新たな公式を検索するようにしている。

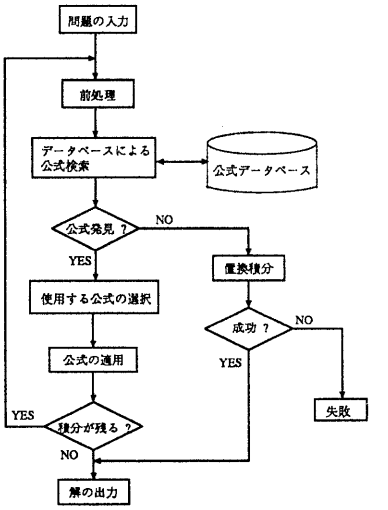

図 1: 不定積分求解の処理のながれ

図の中の前処理の部分では、項の分割と定数のくくりだしを行なっている。この処理は必ず必要である。なぜならば、データベース中の公式は単項式である上に、式全体にかかる定数を記述していないので、この処理を行っておかないとマッチングが失敗する可能性があるからである。

公式が検索されないときには、置換積分を試みるようにしている。これは、変数の変換を行なって被積分関数をより簡単な形に変形し、その関数

に対してデータベース検索を行なうものである。従って、図の中では示していないが、この部分では被積分関数を簡単な形に変形した後、全体の処理を再帰的に呼び出している。

4.2 置換積分

基本的考察

実際の問題においては、それにそのまま該当する公式が存在するとは限らない。そのような場合には、公式が適用できるように式を変形する必要がある。その1つの方法が置換積分である。

置換積分を使うと、積分公式データベースの適用範囲が広がり、公式で解が得られる不定積分の数も増えてくる。

本システムは置換積分の機能を持っており、データベースから公式が見つからない場合には自動的に実行するようになっている。その処理の中では、システムが式の中から別の変数に置き換える部分式を見つけ出し、式を変形した後で新しい変数に対する積分を解いている。従って、ユーザは置換積分について特に意識する必要はない。

derivative–and–divide

本システムで適用した derivative–and–divide[4] という置換積分の方法について考える。被積分関数が2つの関数 $f(x)$ と $v(x)$ の積であるとき、これが次の形ならば、

$$f(x)v(x) = c \cdot f(u(x))u'(x) \quad (3)$$

変数を x から u に変えて、

$$\int f(x)v(x)dx = c \cdot \int f(u)du \quad (4)$$

となる。通常、変数 x としての $f(u(x))$ よりも、変数 u としての $f(u)$ の方が簡単な形であるから、式(4)の左辺よりも、右辺の積分の方が求めやすい。この方法を、被積分関数がn個の関数の積で表される場合に拡張して、システムに採用することにした。

実際に置換積分を実現するために、derivative–and–divide が可能なケースを2つ考えることにした。つまり、被積分関数が $F(x) = f_1(x) \cdots f_n(x)$ であるとき、

$$F(x) = c \cdot u(x) \cdot f_1(u(x)) \cdots f_{n-2}(u(x)) \cdot u'(x) \quad (5)$$

$$F(x) = c \cdot f_1(u(x)) \cdots f_{n-1}(u(x)) \cdot u'(x) \quad (6)$$

のいずれかの形をしているならば、それぞれ次の不定積分に変形できる。

$$\int F(x)dx = c \cdot \int u \cdot f_1(u) \cdots f_{n-2}(u)\,du \quad (5)'$$

$$\int F(x)dx = c \cdot \int f_1(u) \cdots f_{n-1}(u)\,du \quad (6)'$$

この2つのケースの可能性を調べることによって、被積分関数に対して置換積分が可能かどうかを判断している。

この方法において重要なことは、部分式の取り出しと新しい変数への置き換えである。この2つの能力によって置換積分の能力は決定付けられる。現在のところ、部分式としては、積分変数の多項式とその部分式に限定している。この部分を拡張することにより、機能を向上させることが可能となる。

5 システムの評価

ここでは、実際の問題に対して本システムを使用した結果について述べる。システムはGAL92–4上で動作し、計算機はSparcステーションを使用した。

5.1 不定積分求解の評価

ここでは、本システムの不定積分の求解能力について調べた結果を示す。現実的な不定積分に対して評価を行なうために、問題集[6, 7]の中から257個の不定積分を選んでテストした。テストは、従来の方法による求解(解析的手法)と公式を使う方法の特徴を把握するために、以下に示す5種類

表 1: 問題集を使ったシステムの評価 (個)

実行形態	成功	失敗
(1) Reduce	193 (75%)	64
(2) GAL (exp-mode)	167 (65%)	90
(3) GAL (unexp-mode)	160 (62%)	97
(4) GAL (exp&unexp-mode)	173 (67%)	84
(5) Reduce & GAL	238 (93%)	19

の実行形態を用いることにした。テストの結果は表 1 のようになった。

Reduce を最初に用い、解けない問題を本システムで実行するという方法をとると、Reduce 単独では75%だった解答率を93%まで向上させることができた。解析的手法で解けなかった 64 個の問題のうち、45 個 (70%) は公式を使うと解が得られたのである。

6 おわりに

本研究では、積分公式を使用して不定積分を解くためのシステムを開発した。そのために、積分公式を検索するために必要であった数式のパターンマッチャを含んだ公式データベースの検索部を作成した。そして、自動運用が可能な積分公式データベースを構築するため、図書館情報大学で行なわれた設計を一部変更し、新たにデータベースを構築した。

また、公式の運用については、積分公式データベースを自動運用するシステムのプロトタイプを作成した。積分公式の適用範囲を広くするために備え付けた置換積分の機能は完璧ではないが十分に機能している。我々は、このシステムが従来の積分システムの能力をさらに高めるための有効な手段の 1 つになると考えている。今後の課題として、従来のシステムがほとんど対応できない特殊関数を扱えるようにすること、公式選択の際に問題となる数式の表示を見やすくすること、そして分散処理などによる処理の効率化などがある。

参考文献

[1] 佐々木健昭、元吉文男. 国産数式処理システム GAL （デモ用資料）. 数式処理通信. Vol.4,No.1,p6-18(1986).

[2] 三枝義典 . 数式処理システム GAL における数学公式データベースの研究・開発. 図書館情報大学修士論文、1989、茨城、51p

[3] 佐々木健昭、石川正. 数式処理システム GAL におけるパッケージ・プログラミング. version 92.3 , 1992

[4] 下地貞夫. 数式処理システム (基礎情報工学シリーズ 13). 森北出版, 1991

[5] 森口繁一、宇田川佳久、一松信. 数学公式 1, 微分積分・平面曲線. 岩波書店,1956, 東京,318p.

[6] 田代嘉宏. 新·高専の数学 2 問題集, 森北出版,1982

[7] 田代嘉宏. 新·高専の数学 3 問題集, 森北出版,1983