数理解析研究所講究録
第2022巻 2017年 185-191

代数的無理数の計算練習ウェブサイトへの実装について

大阪産業大学・経済学部　服部　純典 (Yoshinori Hattori)
Faculty of Economics,
Osaka Sangyo University

1　はじめに

数学の講義科目に対し、計算演習の補助のためのウェブサイトを構築し、学習者の適切な演習を可能にするプロジェクトを進行している。これは学習達成度の向上に寄与することが期待され、実際既に作成した部分での利用でそのことが確認されている。

このプロジェクトではウェブページで演習問題を提供するのであるが、CGI技術を用いることで、適切な問題を自動生成することにより効果的な演習が可能になっている。前回の研究集会で報告したように、まず最初に取り上げる演習のテーマとして線形代数学の基本変形の基礎を習得してもらうサイトを構築した [1]。このサイトでは、正方行列の階数と行列式の計算練習をすることで基本変形の仕組みを学習する。したがって、出題の行列成分は簡単な整数に限って取り扱うようにしている。その実現のためサーバー側の処理に有理数をそのまま四則演算する機能を実装するした。特にこのサイトでは、学習者の理解を助けるために計算過程の基本変形が1段階ずつ表示できるようにしてあり、それを辿ることで解答の道筋の理解が深まる。つまり、途中計算のプロセスを筆算と同じように示されるので理解の不十分なところを補われ学習効果が上がる。このように有理数計算の実装は必要不可欠なものになっている。

線形代数学にはさまざまな重要な計算プロセスがある。基本変形の応用で計算できる掃き出し法などもあるが、固有値問題は別の計算プロセスが必要となってくるので演習に取り上げることが重要である。この固有値問題では固有値を求めるために代数方程式を解くことになり、その解にはしばしば代数的無理数が登場する。もちろん代数的無理数を避けた問題も可能ではあるが、演習として貧弱となり学習には望ましくない。したがって、開発しているウェブサイトでも代数的無理数の具体的計算、つまり、学習者が紙に書いて計算するような計算を実現する必要が出てくる。

本研究では、代数的無理数の典型例として、有理数の平方根を有理数の係数で線形結合したものについて3タイプの実装を考え、その実装の効率性を消費記憶量と処理時間の両面から実験し、考察することより望ましい実装方法を検討することを目的とした。いままで行われてきた研究では代数的無理数の陽な実装についてはあまりなく、参考にできる研究が見当たらなかった。よってこのような処理系を今後構築する上で本研究は参考になると思われる。

2 実験データの定義と実装方法の提案

2.1 実験データの定義

システムで扱うことを想定している代数的無理数の典型として、いくつかの有理数の平方根に有理数をかけたものを線形結合した無理数を扱うのであるが、その標準形を定義する必要がある。その定義の意義は、表現された数の同値の判定が、その表現の一意性から保証されることになり、システムで重要な意味を持つことになる。

有理数の平方根は分母を有理化したうえで分子の根号内の平方数のくくり出しにより、整数の平方根と既約分数の有理数の積で一意に表される。根号内は異なる素数の積で表され、この根号内の整数を以後**根号整数**と呼ぶ。また、平方根にかかる有理数や単独の有理数のことを**有理係数**と呼ぶ。よって標準形の定義は次のようになる。

$$\frac{a_0}{b_0}+\sum_{i=1}^{N}\frac{a_i}{b_i}\sqrt{p_i}$$

ここで、a_i は非零整数、b_i は非負整数で、a_i と b_i は互いに素である。根号整数 p_i は異なった素数の積で $1<p_i<p_{i+1}$ ある。有理数は $N=0$ と解釈し、平方根の項がない場合である。特に 0 は $a_0=0$、$b_0=1$ と定義する。

この研究の実験は有理数と異なる2つの平方根に有理数のかかったものの和を対象とする。それが実際の筆算で登場する数の事例にあたることで、実験として適当なものとなるからである。実際の実験データは次のようになる。

$$\frac{a_0}{b_0}+\frac{a_1}{b_1}\sqrt{p_1}+\frac{a_2}{b_2}\sqrt{p_2}$$

ここで、根号整数 p_1、p_2 は異なる素数の積、つまり平方数を約数に持たない正の整数で、$|p_i|<20$ かつ $p_1<p_2$ とし、有理係数は既約分数で分母分子は1桁の整数で、符号は分子に持たせる。ゼロは実験に加えても意味がないので今回の実験では除外した。

2.2 実装方法の提案

前節で定義した無理数データを実装することを見ていくことにしたい。その基本となる実装方法には次の3つの方法が提案できる。特に前回の報告でのサイトの有理数の実装と互換性を保つことを念頭に考えていく。

2.2.1 実装方法タイプ1

最初に提案する実装方法は、有理数の部分は以前のデータ表現をそのまま用いて分子と分母を文字 `v` で結合し、それにかかる根号整数を文字 `r` の後に続ける。有理数のみの場合は平方根を $\sqrt{1}$ と見て表現する。いくつかの項の和になっているときは各項の根号整数について昇順にソートして並べて、各項の表現を文字 `a` で結合し全体を表現する。例えば、次のようになる。

$$-\frac{1}{4}-\frac{\sqrt{2}}{8}-\frac{\sqrt{6}}{4} = \frac{-1}{4}\sqrt{1}+\frac{-1}{8}\sqrt{2}+\frac{-1}{4}\sqrt{6}$$
$$\Rightarrow \texttt{-1v4r1a-1v8r2a-1v4r6}$$

2.2.2 実装方法タイプ2

次は各項の有理係数の分母を通分して共通化し、分子に整数と平方根の積の和として表現するもの考える。分子の表現はタイプ1の有理数となっているところが整数になったものである。タイプ1で用いた同じ例で表すと次のようなになる。

$$-\frac{1}{4}-\frac{\sqrt{2}}{8}-\frac{\sqrt{6}}{4} = \frac{(-2)\times\sqrt{1}+(-1)\times\sqrt{2}+(-2)\times\sqrt{6}}{8}$$
$$\Rightarrow \texttt{-2r1a-1r2a-2r6v8}$$

2.2.3 実装方法タイプ3

今度は根号整数を素数の積と見ることで、それに注目して表現する。まず、すべての平方根内に現れる素数を集めて昇順に並べる。これを**素数ベース**と呼ぶことにする。前の例では2と3になる。次に平方根にかかる有理係数を順に並べる。係数に対応する平方根の根号整数の約数である素数を1に、約数でない素数を0として2進数の桁の数を構成する。このとき、最小の素数を最下位の桁として順に上位に並べておく。この2進数を係数の番号とし、この番号を**係数インデックス**と呼ぶこととする。例に則して見ると$\sqrt{6}$は$6=3^1\times2^1$となり$3(=[11]_2)$番目となる。なお最初の0番目は有理数の項となる。有理係数がない（係数が0の）項は空文字として文字aで有理係数を結合し、また素数の積を昇順に文字tで結合する。それらを文字rで結ぶ。なお、後部の有理係数に0が続くときは省略するものとする。同じ例で見ると次のようになる。

$$-\frac{1}{4}-\frac{\sqrt{2}}{8}-\frac{\sqrt{6}}{4} = \frac{-1}{4}\sqrt{3^0\cdot2^0}+\frac{-1}{8}\sqrt{3^0\cdot2^1}+\frac{0}{1}\sqrt{3^1\cdot2^0}+\frac{-1}{4}\sqrt{3^1\cdot2^1}$$
$$\Rightarrow \texttt{-1r1v4a-1v8aa-1v4r2t3}$$

3 比較実験の方法

実験データは10,000個をランダムに用意し、同じ数値データを3つの実装形式で表現する。3つの演算として和、積、逆数についてその計算時間と計算結果の表現データ長の平均で評価することにした。10,000個のデータを処理した時間を記録し、それを100回繰り返し、その処理時間データを統計処理することとした。また、演算アルゴリズムの検証も兼ねて10,000個の各々の表現の計算結果を記録し、その結果データを表現変換によってすべて一致することを調べ、問題がないことを確認した。そのことにより計算アルゴリズムが正確であることが保証されたと判断した。

実験は処理プログラムの能力をできるだけ純粋に評価できるように、スタンドアローンのマシンに http サーバープログラム (Apache) を起動して、ウェブ・ブラウザから localhost にアクセスするようにして行い、処理プログラムはサーバープログラムから CGI でデータを受け取るようにした。プログラムのコーディングには実際の実装に用いるプログラミング言語 Perl を用いた。では、実験材料となるデータと演算の方法の詳細について見ていく。

3.1 実験データについて

2節で述べたように有理数と2つの平方根の項の和で表されたデータを扱う。有理係数は分母・分子ともに絶対値が10以下の整数にし、根号整数は20以下の整数で、乱数を発生して生成し、そのあと標準形に整形した数値データを10,000個用意した。実験の比較をするために同じデータを3種の表現に変換し、各々別のファイルにテキストデータで保存しておいた。

3.2 和の演算実験について

データファイルからデータを順に読み出し、次のデータとの和を計算することを順次行う。最後のデータは最初のデータとの和を計算することで10,000回の計算を行う。計算のアルゴリズムについては、タイプ1と2は基本的に有理係数を根号整数についてマージすることで計算が実現できる。しかし、タイプ3に対しては多少複雑となる。まず、素数ベースをマージする。その結果から、有理係数の係数インデックスを変換しなければならない。その上で有理係数のマージを行うこととなる。したがって、タイプ3は計算時間がかかることが予想される。

3.3 積の演算実験について

和の場合と同じように隣り合うデータとの積を計算する。アルゴリズムは和の場合と違い、少々複雑となる。タイプ1と2は平方根の積で平方数をくくり出す必要がでてくる。それは積をとる2つの根号整数の最大公約数がくくり出す数となるので、ユークリッドの互除法でそれを算出する。すべての項の積を計算してそれらをソートすることで計算結果を得る。処理効率の観点からは計算の中でタイプ1は有理数計算が必要となるが、タイプ2は分子の積となり、整数の計算できるので計算の効率的にはやや有利と思われる。ただ、タイプ2は共通分母の形を採るので、扱う数値の桁数が大きくなる傾向があることも予想される。

一方、タイプ3では有理係数の係数インデックスで処理をすることとなる。和の時と同じように素数ベースをマージし、係数インデックスを変換する。積の係数インデックスはインデックスの排他的論理和で求まり、くくり出す数は係数インデックスの論理積で求まる。すべての項の積について同様の計算を行った後、素数ベースのうち関係する係数がない素数があれば除外する処理が必要になる。除外処理では素数ベースから除くだけでなく、有理係数の係数インデックスを変換することとなる。

3.4 逆数の演算実験について

この計算は前の2つの計算と違い各数値データ自身の逆数を計算することで実現できる。しかし、計算のアルゴリズムは複雑になる。いわゆる分母の有理化の計算となり、根号整数に含まれる素数の数だけ計算を繰り返す必要がある。

まず一般的なアルゴリズムを考える。分母の根号整数に含まれる素数の1つを q とする。分母を $p_1+p_2\sqrt{q}$ とおく。ここで、p_1、p_2 は $\sqrt{q}$ は含まないが他の平方根は含む数とする。分母と分子に $p_1-p_2\sqrt{q}$ をかけることで分母が

$$(p_1+p_2\sqrt{q})(p_1-p_2\sqrt{q})=p_1^2-p_2^2q$$

となり、分母から $\sqrt{q}$ の要素がなくなる。これを分母から平方根の項がなくなるまで行うので、繰り返しの最大回数はタイプ3で言うところの素数ベースの個数となる。

タイプ1のアルゴリズムはまず有理係数を通分してタイプ2に変換することから始める。なぜなら、そのままの計算では有理数の計算が多数回発生し非常に計算時間がかかるからである。実際今回のデータでの計算ではかなりの差が出た。その処理のあとタイプ2のアルゴリズムに合流する。タイプ2のアルゴリズムでは分母の中で根号整数の1つの素因数を r とすると、$\sqrt{r}$ を積として含む項の符号を反転した数を分母と分子の両方にかける。これを分母が整数になるまで繰り返すのである。最後にタイプ1では各有理係数に分母を分割して約分を行ってタイプ1の形式に戻す。タイプ3も同じようなアルゴリズムであるが、分母と分子への乗数を求める時に係数インデックスのビットで符号の反転が判断できる。

いずれのタイプの計算でも共通するアルゴリズムにおいて高速化の工夫が必要である。分母と分子の係数を1つの固定の配列で計算するより、それぞれ2つの配列を用意する。つまり、多重配列、例えば `$b[0][0]`～`$b[0][$N]` と `$b[1][0]`～`$b[1][$N]` を用意して乗算を1回するごとに `$b[0][$i]` と `$b[1][$i]` を切り替えて行う。これにより計算結果を元の配列に戻す操作が省略できる。

また、乗算計算でも高速化の工夫をする。まず分母の係数の符号反転作業を先に行い、それを分子と乗算し、その後符号反転した分母で分母の乗算計算をする。これは分母の乗算計算は符号反転前でも後でも結果が変わらないためである。さらに、分母の乗算計算でも工夫をする。項の自身の自乗は符号反転の要素がある項は符号を反転する。また違う項との積は符号反転要素のある項どうしの計算とない項どうしの計算のみを行う。打ち消すことがわかっている項の計算を省略するためである。

4 実験結果

演算別に実験結果を示す。まず最初に同じ数のデータ長の違いを見てみる。

4.1 元データのデータ長

用意した10,000個のデータの長さの平均値を算出したものを次に示す。

データ型	タイプ1	タイプ2	タイプ3
平均長 (byte)	19.667	19.347	19.387

4.2 和の演算時間

各タイプデータ10,000個に対する和の計算の総合処理時間を記録し、その100回分の記録データから平均時間と計算結果の表現データ長を次に示す。

データ型	タイプ1	タイプ2	タイプ3
平均 (sec.)	1.191	1.574	2.848
標準偏差 (sec.)	0.150	0.189	0.438
最大値 (sec.)	1.203	1.610	2.937
最小値 (sec.)	1.172	1.562	2.828
平均長 (byte)	33.043	33.375	36.247

4.3 積の演算時間

各タイプデータ10,000個に対する積の計算の総合処理時間を記録し、その100回分の記録データから平均時間と計算結果の表現データ長を次に示す。

データ型	タイプ1	タイプ2	タイプ3
平均 (sec.)	5.512	4.325	8.967
標準偏差 (sec.)	0.624	0.735	1.346
最大値 (sec.)	5.594	4.438	9.125
最小値 (sec.)	5.453	4.281	8.828
平均長 (byte)	66.919	65.525	59.693

4.4 逆数の演算時間

各タイプデータ10,000個に対する逆数の計算の総合処理時間を記録し、その100回分の記録データから平均時間と計算結果の表現データ長を次に示す。

データ型	タイプ1	タイプ2	タイプ3
平均 (sec.)	7.956	6.250	13.991
標準偏差 (sec.)	1.117	1.009	1.873
最大値 (sec.)	8.109	6.438	14.219
最小値 (sec.)	7.828	6.203	13.782
平均長 (byte)	75.980	50.388	73.062

5 結果の考察と検討

5.1 演算の計算時間について

まず計算時間の検討である。和の計算では確かにタイプ1が最速であるが、他の2種の演算ではタイプ2が最速となっている。これはタイプ1が有理数計算が多くなると不

利となるのが原因である。和に比べて積の計算ははるかに有理数計算の回数が多いことから理解できる。逆数の計算は積の計算の繰り返しなのでさらに時間がかかる。タイプ3はいずれの計算も時間がかかっているが、和の計算時間に対して逆数の計算時間の増加率は約5倍でタイプ1の7倍以上より優れていると言える。

また、全体の処理について言えることであるが、同じアルゴリズムでもコーディングの方法で処理時間がかなり変わることが分かった。具体的に、プログラミング言語 Perl に則して述べる。100 回の繰り返し処理について考えると、例えば、

```
foreach $i (0..99){ ··· }
```

とコーディングするのと

```
for($i=0,$i<100;$i++){ ··· }
```

とするのでは、後者が前者に比べて1割ほど処理時間が早くなった。同じ処理をするのにも、実時間での処理を前提に考える場合、このようなことを検証する必要があることが実際に示された。

5.2 データ長について

比較的簡単なデータである元データではタイプ2と3がほぼ同じでタイプ1は1%長いだけでほとんど変わらない。しかし、積の計算結果ではタイプ1はタイプ3に比べ12%長いことになっている。平方根の項が増えてくるとタイプ3の表現の方が有利である。さらに、逆数の計算結果は最短のタイプ2に比べて最長のタイプ1は約1.5倍になっている。これは有理数の分母が大きくなるとまとめた方が表現が短くて済むというように理解できる。

5.3 総合的評価

演算時間と表現データ長の両方を勘案するとタイプ2が最良であると考えられる。共通分母により表現データ長が長くなるような先入観があったが意外とそうではない結果である。

6 今後の展望

この実験結果を受けて、最良と判断されたタイプ2の表現形式を実装して線形代数学の固有値問題を扱えるよう演習サイトの拡張を試みる。さらにより基本的な数学の演習として代数方程式の解法や分母の有理化計算をシステムとして構築できるので応用を考えたい。

参考文献

[1] 服部純典, "線形代数学の計算ドリルサイトの目的と実装", 京都大学数理解析研究所講究録, No.1978(2015), pp.117-124.