JAPLA研究会資料　2019/3/9

Ｊでピタゴラス数(２つの平方数の和に分解)を求めるーその２
自然数に対するピタゴラス数

西川　利男

１．シルバーマンの攻略法とは

先のJAPLAの例会では、素数についてのピタゴラス数分解(２つの平方数の和に分解)のＪプログラムによる計算を報告した[1]。

シルバーマンによれば[2]、「一般的な課題に取り組む前に、問題を分割して、まずは易しい特別な例について攻略する。その手法を少しずつ統合して、最終的に一般の問題にもっていくのが、大切である。」と言っている。私もそのとおりだと思う。

[1] 西川利男、「Ｊでピタゴラス数(２つの平方数に分解)を求めるー
　　Ｊによるフェルマーの無限降下法」JAPLA研究会資料 2019/1/19
[2] ジョセフH. シルバーマン、鈴木治朗訳「はじめての数論」
　　ピアソンエデュケーション（2001).

今の場合でいえば、素数に対しての２つの平方和への分解は前回の報告のようにすでに出来ているので、これをもとに自然数についても出来るように、統合していくことが今回の目標となる。

・分割　対象とする自然数を素数の積に分解する
・攻略　それぞれについて、平方和分解を行う。
・統合　前回、用いた次の恒等式により積から和の形に直す

$$(u\text{^}2+v\text{^}2)*(A\text{^}2+B\text{^}2)=(u*A+v*B)\text{^}2+(v*A-u*B)\text{^}2$$

として、最終的に自然数のピタゴラス数分解のシステムを構築する。

また、前回も述べたが素数には、２種類あって、

素数　4n+1 型素数　＝＞　分解可能
　　4n+3 型素数　＝＞　分解不可能

と、別々に扱わなくてはならない。

今回も、原理を示した後、数式でというよりＪのプログラミングを通して、具体的に、実際例で説明したいと思う。

２．簡単な例

まずは、簡単な小さな数で見てみよう。

65 は次のように素数に分解できる。（素因数分解）

65 = 5 * 13

そして、5 と 13　は前回示したように、次の２乗の和で表せる。

5 = 2^2 + 1^2

13 = 3^2 + 2^2

したがって

65 = (2^2 + 1^2) * (3^2 + 2^2)

となる。ここで、例の積から和への変換の恒等式を適用する。

65 = (2*3 + 1*2)^2 + (2*2 - 1*3)^2

= (6 + 2)^2 + (4 - 3)^2

つまり

65 = 8^2 + 1^2

となって、２つの平方数の和で表せた。

それでは、12　ではどうだろうか。

12 = 2 * 2 * 3

この場合は、素因数分解で 3 が出てきた。3 は 4n+3 型の素数であり、２つの平方数の和ではあらわせない。したがって元の 12 も平方数の和に分解できない。

３．もう少し複雑な例

m = 1105 を２つの平方数の和に分解する。

(1) m を素因数分解する。

m = 5 * 13 * 17

(2) それぞれの素数を平方数の和で表す。

5 = 2^2 + 1^2

13 = 3^2 + 1^2

17 = 4^2 + 1^2

(3) 恒等式を繰り返し用いて、積を和への変換を繰り返す。

m = 1105

= 5 * 13 * 17

= (2^2 + 1^2) * (3^2 + 2^2) * 17

= (2*3 + 1*2)^2 + (2*2 - 1*3)^2 * 17

= (6 + 2)^2 + (4 - 3)^2 * 17

= (8^2 + 1^2) * (4^2 + 1^2)

```
= (8*4 + 1*1)^2 + (8*1 - 1*4)^2
= (32 + 1)^2    + (8 - 4)^2
=  33^2  + 4^2
```

つまり

```
1105 = 33^2 + 4^2
```

と２つの平方数の和に分解できた。

４．Ｊのプログラム

全体のプログラム run は

前処理(素因数の種類分けなど) run0

積から和の繰り返し処理 run1

とに分けて作成した。

具体的にＪのコーディングに入る前に、私なりにプログラミングはどうあったらよいかについて、いささか述べたいと思う。

実は、素因数の種類分けなどで、思った以上にてこずったのである。素因数の集合から、指定した数を取り除こうとしたとき、当然Ｊのプリミティブにあるものと思っていたがそうではなかった。そのための動詞をつくったりした。

また、積から和の繰り返し処理では、二つずつ取り上げ、処理しその結果を次にというところで、単純なループではうまくいかなかった。

Ｊのコンパクトなプログラムという特徴も発揮できず、何とか動かすためにはまことにみっともないコーディングになってしまった。

プログラミングとは、数学などの論理ではない。計算処理を操作する文章部分として大切である。かっこいい言い方をすればアルゴリズムである。

言うなれば、プログラミングとは文章を綴る作業である。そしてこれは一種の職人芸的なわざ、技術であると私には感じられた。

この後述べる run, run0, run1 の最終的なコーディングにいたる前に 、test0, test1,..test5 と途中経過を見ながらエラーを直しつつ、手直し作業を何べんやったことか。すっかり楽しませて(?)くれたのである。

４．１　run1 のプログラムとその実行

先に、こちらの処理について述べる。素因数分解で得られた素数の集合をもとに、積から和の変換処理(プログラム mul2sum)を繰り返す。

先の m = 1105 について、途中経過を示しつつ、プログラムを説明する。
あらかじめ、J のプリミティブ q:　を用いて、素数の集合を求めておく。

```
   q: 1105
5 13 17
```

実行のようすは途中経過を示しつつ、次のようになる。

```
   run1 5 13 17
# of R: 3
*** mult to sum process ***
k:0
ta:5
+---+
|2 1|
+---+
k:1
13
+---+
|3 2|
+---+
+---+---+
|2 1|3 2|
+---+---+
tc = 8 1
65
---Enter CR ------

k: 2
k{R = 17
td = 4 1
+---+---+
|8 1|4 1|
+---+---+
5 13 17 = 33^2 + 4^2
```

先に示した手計算の処理を、そのまま、コーディングしたままである。
素数の集合から、２つ取りそれについて、積から和の変換処理(mul2sum)を行い、その結果に次の素数と変換処理(mul2sum)を行う、…　ということを続ける。
通常なら繰り返しループを用いるところが、うまくいかなかった。

Ｊのコーディングは次のようになる。サブルーティンとして、ferm, mul2sum, classify, inwait を使用。途中経過も表示。

```
run1 =: 3 : 0
R =. y.
if. (3 e. 4|R)
  do.
     wr '4n+3 type => not decomposed !'
     return.
  end.
QQ =. |: (classify Q) ,: (~. Q)
wr '# of R: ', ":#R
wr '*** mult to sum process ***'
k =. 0
    wr 'k:', ":k
    ta =. k{R
    wr 'ta:', ": ta
    wr < ferm ta
k =. k + 1
    wr 'k:', ": k
    tb =. (k){R
    wr tb =. (k) {R
    wr < ferm tb
    wr (< ferm ta), (< ferm tb)
    tcc =. (ferm ta) mul2sum (ferm tb)
    wr 'tc = ', (": tcc)
    tc =. (*: 0{tcc) + (*: 1{tcc)
    wr tc
if. (2 = #R) do.
    tcc
    return.
```

```
    end.

label_again.
inwait ''
k =. k + 1
    wr 'k: ', (": k)
    wr 'k{R = ', (": k{R)
    td =. ferm k{R
    wr 'td = ', (": td)
    wr tcc;td
    tt =. tcc mul2sum td
if. k < >: # R
      do.
         wr (": y.),' = ', (": 0{tt), '^2 + ', (": 1{tt), '^2'
         goto_fin.
      end.
         tcc =. tt
         goto_again.
label_fin.
tt
)

classify =: 3 : '+/"(1) (~. y.) =/ y.'

NB. convert mult to sum equation ========================
mul2sum =: 3 : 0
:
'u v' =. x.
'A B' =. y.
((u*A) + (v*B)), ((u*B) - (v*A))
)

wr =: 1!:2&2
rd =: 1!:1
```

４．２　run0 のプログラムとその実行

素因数分解により、因数が複数あるときは、２乗としてくくり出せて、その中に 4n+3 型素数が含まれていても平方和分解が可能となる。

具体例をあげて、説明しよう。

m = 25798500 とかなり大きい数で、なお、3, 5, 7 のように 4n+3 型の素因数があるが、これらをくくり出す前処理が run0 である。

```
   run0 25798500
2 2 3 3 3 3 5 5 5 7 7 13
2  2
4  3
3  5
2  7
1 13
-- prime classify --
+----+------------------+
|5 13|7 7 5 5 3 3 3 3 2 2|
+----+------------------+

run0 =: 3 : 0
wr Q =: q: y.      NB. global
QU =. ~. Q
wr QQ =. |: (classify Q) ,: (~. Q)
B =: (2 <: 0{"(1) QQ) # QU

wr '-- prime classify --'
QA =. Q;''
i =. 0
while. i < ({. $ QQ)
  do.
    'n m' =. (i{QQ)
    nn =. (+: <. -: n)
    j =. 0
    while. j < nn
       do.
        QA =. QA move m
```

```
          j =. j + 1
     end.
      i =. i + 1
  end.
QA
)
```

ここで、サブルーティンmoveは複数の集合から指定した数を１つだけ取り除く関数である。Jのプリミティブ -. ではすべてとり除かれてしまう。
この目的のため、move, excl次のように定義したが、なかなか大変であった。

```
NB. exclude element
NB. 2 2 3 3 3 excl 2 => 2 3 3 3 : exclude 2
NB. 2 2 3 3 3 excl 3 => 2 2 3 3 : exclude 3
excl =: 3 : 0
:
data =. x.
m =. (#data) # 0
ex =. y.
(-. (1 (data i. ex) } m)) # data
)

NB. move
move =: 3 : 0
:
'left right' =. x.
ex =. y.
(left excl ex);(ex, right)
)
```

４．３　runのプログラム

まず、run0 を、次に run1を行うことによって、最終プログラムrunとした。

```
run =: 3 : 0  NB. renamed and compacted from test5
'R S' =. run0 y.
if. 0 = #S
 do.
   TT =. run1 R
```

```
   wr (": y.), ' = ',  (": 0{TT),'^2 + ', (": 1{TT),'^2'
 else.
   TT =. run1 R
   SS =. %: */S
   wr ST =. SS * TT
   wr (": y.), ' = ',  (": 0{ST),'^2 + ', (": 1{ST),'^2'
 end.
'** end **'
)
```

５．その他の実行例

シルバーマンの本から、いくつかの練習問題をやってみた。[2] p.176

```
   run 1885
5 13 29
1  5
1 13
1 29
  ---
+---+---+
|2 1|3 2|
+---+---+
  ---
+---+---+
|8 1|5 2|
+---+---+

1885 = 42^2 + 11^2
** end **
```

素因数はすべて 4n+1 型素数なので、平方和分解ができる。

```
   run 3185
5 7 7 13
1  5
2  7
1 13
+---+---+
```

```
|2 1|3 2|
+---+---+
3185 = 56^2 + 7^2
** end **
```
この問題は、4n+3 型素数を含むが 2 個なので、平方和分解ができた。

```
   run 4370
2 5 19 23
4n+3 type => not decomposed !
** end **
```
この問題は、4n+3 型素数が 19 と 23 の 1 個ずつなので、平方和分解は出来ない。

```
   run 25798500
2 2 3 3 3 3 5 5 5 7 7 13
2  2
4  3
3  5
2  7
1 13
+---+---+
|2 1|3 2|
+---+---+
25798500 = 5040^2 + 630^2
** end **
```

６．おわりに

ピタゴラス数の J プログラムで、前回の発表のとおり素数に対しては完成していたので[1]、今回こんなにてこずるとは思ってもみなかった.

ごらんのように、J のプログラムとしては、まことにみっともないものとなってしまった。J の特徴である配列処理、関数型言語、再帰処理などがうまく生かせなかった。

しかし、実用に耐えるプログラムとはかならずしもエレガントな詩的ものではなく、文章を綴るようなダサイ散文的なものに私はなると思う。

プログラムの記述性について、J がよく耐えてくれ、有効に働いてくれたことに、あらためて感謝したい。

プログラムリスト

```
run0 =: 3 : 0
wr Q =: q: y.      NB. global
QU =. ~. Q
wr QQ =. |: (classify Q) ,: (~. Q)
NB. ({."(1) (0&=)@(2&|) QQ) # ~. Q
B =: (2 <: 0{"(1) QQ) # QU
wr '-- prime classify --'
QA =. Q;''
i =. 0
while. i < ({. $ QQ)
  do.
NB.     wr '(i:', (": i), ') ', ": (i{QQ)
    'n m' =. (i{QQ)
    nn =. (+: <. -: n)
NB.     wr n, nn, m
    j =. 0
    while. j < nn
       do.
NB.         wr 'test ', ": j
        QA =. QA move m
NB.          wr QA =. QA move m
NB. inwait ''
         j =. j + 1
    end.
     i =. i + 1
  end.
QA
)
run1 =: 3 : 0
R =. y.
if. (3 e. 4|R)
  do.
     wr '4n+3 type => not decomposed !'
     return.
  end.
```

```
QQ =. |: (classify Q) ,: (~. Q)
wr '# of R: ', ":#R
wr '*** mult to sum process ***'
k =. 0
    wr 'k:', ":k
    ta =. k{R
    wr 'ta:', ": ta
    wr < ferm ta
k =. k + 1
    wr 'k:', ": k
    tb =. (k){R
    wr tb =. (k) {R
    wr < ferm tb
    wr (< ferm ta), (< ferm tb)
    tcc =. (ferm ta) mul2sum (ferm tb)
    wr 'tc = ', (": tcc)
    tc =. (*: 0{tcc) + (*: 1{tcc)
    wr tc
if. (2 = #R) do.
    tcc
NB.       wr (":y.) ,' = ', (": 0{tcc), '^2 + ', (": 1{tcc), '^2'
NB.       goto_fin.
    return.
    end.
label_again.
inwait ''
k =. k + 1
    wr 'k: ', (": k)
    wr 'k{R = ', (": k{R)
    td =. ferm k{R
    wr 'td = ', (": td)
    wr tcc;td
    tt =. tcc mul2sum td
if. k < >: # R
      do.
         wr (": y.),' = ', (": 0{tt), '^2 + ', (": 1{tt), '^2'
```

```
          goto_fin.
       end.
          tcc =. tt
          goto_again.
label_fin.
tt
)

NB. convert mult to sum equation ========================
mul2sum =: 3 : 0
:
'u v' =. x.
'A B' =. y.
((u*A) + (v*B)), ((u*B) - (v*A))
)
classify =: 3 : '+/"(1) (~. y.) =/ y.'
NB. exclude element
NB. 2 2 3 3 3 excl 2 => 2 3 3 3 : exclude 2
NB. 2 2 3 3 3 excl 3 => 2 2 3 3 : exclude 3
excl =: 3 : 0
:
data =. x.
m =. (#data) # 0
ex =. y.
(-. (1 (data i. ex) } m)) # data
)
NB. move
move =: 3 : 0
:
'left right' =. x.
ex =. y.
(left excl ex);(ex, right)
)
```

```
inwait =: 3 : 0
    wr '---Enter CR ------'
     rd YN                      NB.  キー入力による途中経過表示
     YN =. rd 1
     if. 0 = #YN                NB.  そのまま空CRなら次へ続行
       do. goto_CR.
       else.
       if. 'n' = YN                 NB.  'n'なら実行取りやめ
         do. return. end.           NB.  'y', 'yes'なら次へ続行
     end.
label_CR.
''
)
```