ピタゴラス数は何通りあるか？　そして、どんな数か？－その３
－Beilerの方法によるピタゴラス数の計算―

西川　利男

ピタゴラス数の計算として、前回の方法は斜辺を元に他の２辺を求めるもので、これは平方和への分解という整数論ではかなり難物のテーマであった。

今回はこれとは全く別の方法、すなわち直角をはさむ一辺を与えて他の一辺と斜辺とを求めるものである。これには私のタネ本としているもう一冊の整数論の本 Albert H. Beiler, ″Recreatios in the Theory of Numbers″, Chapter XIV, The Eternal Triangle[1] を参考にしてプログラミングした。

１．ピタゴラス数の基本の性質

ピタゴラスの定理より、直角をはさむ２辺をAとB、斜辺をCとしたとき

$A^2 + B^2 = C^2$

が成り立つ。これを満足する３つの整数の組がピタゴラス数であり、整数mとnとするとき、次の式で与えられえる。

$A = m^2 - n^2,\quad B = 2mn,\quad C = m^2 + n^2$

ここで、Bは偶数であるので、簡単な考察により、Aは奇数、Cも奇数になる。

２．ピタゴラス数の可能な数は何通りあるか？

直角をはさむ一辺(Leg, 足)の長さNが与えられたとき、ピタゴラス数の可能な組は何通りあるか？　既約、非既約を問わず、その数はBeilerにより次の式で計算される。

いま、Nが素因数分解により次のように分解されたとすると

$N = (2\ \hat{}\ a_0)\cdot(p_1\ \hat{}\ a_1)\cdot(p_2\ \hat{}\ a_2)\cdots(p_n\ \hat{}\ a_n)$

ピタゴラス数の可能な数(L)は次式になる。

$$L = \frac{(2a_0 - 1)(2a_1 + 1)(2a_2 + 1)\cdots(2a_n + 1) - 1}{2}$$

たとえば

$N = 60 = 2^2\cdot 3\cdot 5$

となるので

$$L = \frac{(2\cdot 2 - 1)(2\cdot 1 + 1)(2\cdot 1 + 1) - 1}{2} = \frac{3\cdot 3\cdot 3 - 1}{2} = 13$$

一辺が60では可能な数は13通りある。

ピタゴラス数で一辺(足)を与えたときに可能な数は、とても大きな数になる。斜辺からのときの数は0, 1がほとんどで前回の問題がいかに難物だったかがわかる。

なお、Beilerの本にはこれらのピタゴラス数の可能な数を100まで、また100から1000の主要なものにわたって一覧表としてあげてある。

Jによるプログラムは最後にあげた。いくつかの計算例を途中経過（素因数分解）とともにあげた。

```
   fr_leg 60
60 = 2^2*3^1*5^1
Freq. of Pytha Number: 13
   fr_leg 30
30 = 2^1*3^1*5^1
Freq. of Pytha Number: 4
   fr_leg 45
45 = 3^2*5^1
Freq. of Pytha Number: 7
   fr_leg 24
24 = 2^3*3^1
Freq. of Pytha Number: 7
   fr_leg 48
48 = 2^4*3^1
Freq. of Pytha Number: 10
   fr_leg 120
120 = 2^3*3^1*5^1
Freq. of Pytha Number: 22
```

Jのプログラムの前半では、素因数分解の表示を見やすく整えるのにJの正規表現機能を利用した[2]。例えば60のときは次のように変換表示される。

```
   require 'system¥main¥regex.ijs'
   PATP =. '[[:digit:]]+([[:space:]]+)[[:digit:]]+'
   ]QC =: q: 60
2 2 3 5
   ]UC =: ~. QC
2 3 5
   ]PC =: +/"(1) (UC =/ QC)
2 1 1
   ]UPC =. ": ,UC,.PC
2 2 3 1 5 1
   ]UP =. ((PATP;,1);'*') rxrplc ((PATP;,1);'^') rxrplc UPC
2^2*3^1*5^1
```

後半は計算式をそのままプログラムしたので、とくに問題はないだろう。

３．ピタゴラス数の値はどうなるか？

つぎに、いよいよ直角三角形の一辺(Leg 足)を与えてもう一方の辺、斜辺の長さ、つまりピタゴラス数の値を求めていこう。これもBeilerの本のp.108-109に従った。

一辺(Leg 足)の長さAが奇数か、偶数かによって別々の方法になる。

(1) Aが奇数のとき

Aを２つの因数に分ける。Aが素数ならば、1とAとする。

最初に示したピタゴラス数の性質より

$A = m^2 - n^2$で　$C = m^2 + n^2$, $B = 2mn$

である。 これよりAを整数で因数分解して、m, nの組を探し出し、B, Cを決める。

例えば

A ＝ 35　ならば

・　m＋n ＝7,　m－n ＝5 とすると、これを解いて m＝6, n＝1 となり
　　B ＝ 2mn＝2・6・1 ＝12,　C ＝ m^2+n^2＝37　が得られる。

・　m＋n ＝35, m－n ＝1 とすると、これを解いて m＝18, n＝17 となり
　　B ＝ 2mn＝2・35・1 ＝612,　C ＝ m^2+n^2＝613　が得られる。

(2) A が偶数のとき

同様にして A を２つの因数に分けるが、分け方が違う。

例えば

A ＝ 12　ならば、 2mn ＝ 12 であり

・　m＝6, n＝1 とすると
　　B ＝ m^2-n^2＝36－1 ＝35,　C ＝ m^2+n^2＝ 36＋1 ＝37　が得られる。

・　m＝3, n＝2 とすると
　　B ＝ m^2-n^2＝9－4 ＝5,　C ＝ m^2 ＋ n^2＝9＋4 ＝ 13　が得られる。

このようにして得られるピタゴラス数は互いに素の既約ピタゴラス数である。非既約も含めた全てのピタゴラス数を得るには、いろいろな係数 K を掛けたものについて探索を行う。

$K(m^2-n^2)$, $K(2mn)$, $K(m^2+n^2)$

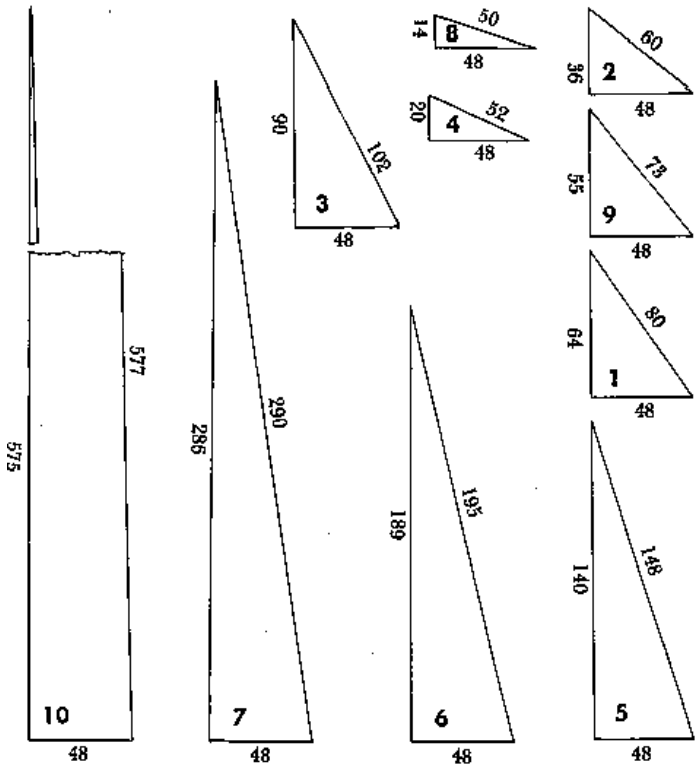
これを分解して因数が素数になるまで、繰り返す。その際、同じものが重複して得られることもあるので、これらを整理して最終結果とする。

J によるプログラムは、最後にあげた。計算の実行例は次のようになる。

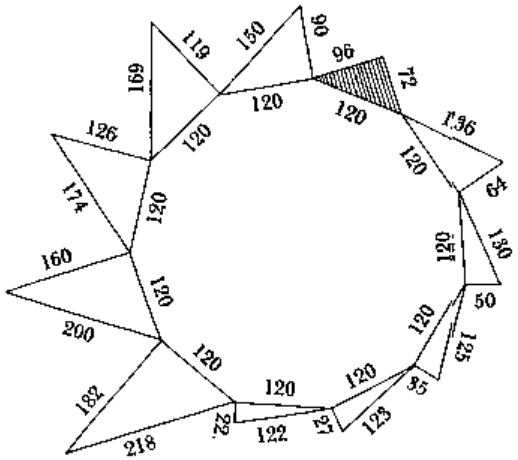
```
 pyt 35
composite
K:1, R:35 1 / 35 612 613
K:1, R: 7 5 / 35  12  37
K:5, R:7 1 / 35 120 125
K:7, R:5 1 / 35 84 91
 pyt 12
composite
K:1, R:6 1 / 12 35 37
K:1, R:3 2 / 12  5 13
K:3, R:2 1 / 12 9 15
K:4, R:3 1 / 12 16 20
```

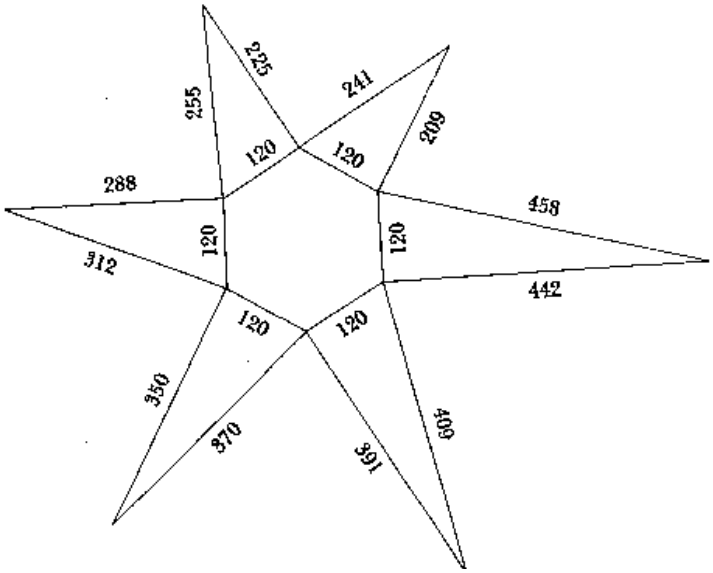
直角三角形の一辺(Leg 足)を元にしたピタゴラス数の可能な例は、大変な数になる。
48 のときは 10 通り、120 では 23 通り、斜辺のとき 1 通りで全部で 24 通りになる。

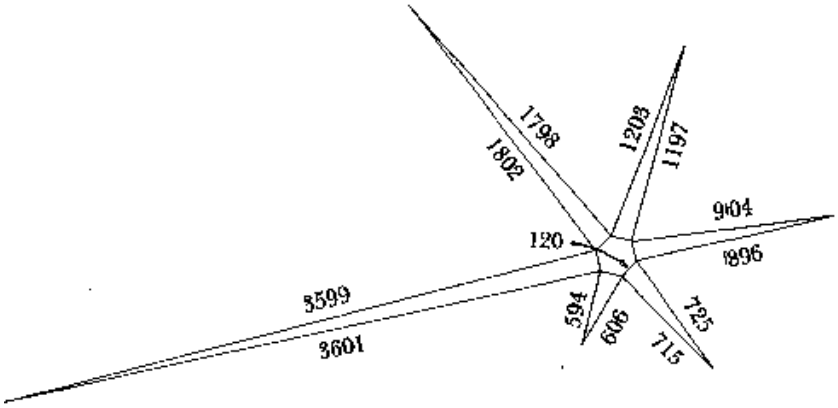
Beiler の本では、これらの例を図で示している。



```
   pyt 48
composite
*** Pythagorean by Leg ***
48 575 577
48 140 148
48  55  73
48  20  52
48 286 290
48  64  80
48  14  50
48 189 195
48  36  60
48  90 102
   pyt 120
composite
*** Pythagorean by Leg ***
120 3599 3601
120  896  904
120  391  409
120  209  241
120  119  169
120   64  136
120 1798 1802
120  442  458
120  182  218
120   22  122
120 1197 1203
120  288  312
120   27  123
120  715  725
120  160  200
120   35  125
120  594  606
120  126  174
120  350  370
120   50  130
120  225  255
120   90  150
```







文献

[1] A. Beiler, "Recreatios in the Theory of Numbers", Dover (1966).

[2] 西川利男「Jの正規表現プログラミングーⅠ，正規表現とはーPerlと比較しつつ，
Jのボックス表示の文字化け解消への利用」J言語研究会資料2004/9/25.

Jのプログラム・リスト

```
NB. Beiler Number Theory
wr =: 1!:2&2
NB. Frquency of Decomposition by Leg
NB.   calculated on Beiler p.116, Formula 2
fr_leg =: 3 : 0
QC =. q: y.
UC =. ~. QC
PC =. +/"(1) (UC =/ QC)
NB. Display Decomp by Regular Expression = 2011/5/4
require 'system¥main¥regex.ijs'
UPC =. ": ,UC,.PC
PATP =. '[[:digit:]]+([[:space:]]+)[[:digit:]]+'
UP =. ((PATP;,1);'*') rxrplc ((PATP;,1);'^') rxrplc UPC
wr (": y.), ' = ', UP
if. 1 = 2 e. UC
 do.
    RA =. (<: +: {. PC)
    RB =. (>: +: }. PC)
 else.
    RA =. 1
    RB =. (>: +: PC)
 end.
'Freq. of Pytha Number: ',": -: <: */RA, RB
)

NB. Chapter XIV The Eternal Triangle p.108-109
NB. Find Pythagorean Number from Leg

quotient =: 3 : 0
y =. y.
if. 0 = 2|y do. y =. -: y end.
N =. >: i. y
P =. y % N
Q =. <. P
A =. (P = Q)#Q
B =. (P = Q)#N
AB =. A,.B
NB. (<. -: {. $AB) {. AB
)

quotientx =: 3 : 0
y =. y.
N =. >: i. y
P =. y % N
```

```
Q =. <. P
A =. (P = Q)#Q
B =. (P = Q)#N
B
)

pyodd =: 3 : 0"(1)
'MM NN' =. y.
M =. -: (MM + NN)
N =. -: (MM - NN)
A =. -/ *: M, N
B =. 2 * M * N
C =. +/ *: M, N
A, B, C
)

pyeven =: 3 : 0"(1)
'M N' =. y.
A =. 2 * M * N
B =. -/ *: M, N
C =. +/ *: M, N
A, B, C
)

NB. Pythagorean by Leg ====================================
NB.   2011/4/27
pyt =: 3 : 0
y =. y.
QC =. q: y.
if. 1 = #QC do. wr 'prime' else. wr 'composite' end.
NB. wr '========'
Q0 =: quotient y.
R0 =. ((-:@#){.]) Q0
NB. wr pyeven  R0
NB. QQ =. 1, 2, 3, 4, 5, 6, 7
QQ =. quotientx y
S =. ''
i =. 0
while. i< (#QQ) do.
NB.  wr 'loop i: ', (":i), ' ==============='
  K =. i{ QQ
  QX =. quotient y % K
  RX =: ((<.@(-:@#)){.]) QX
  if. (0=2|{."(1) RX) do. Res =. K * pyeven RX end.
  if. (1=2|{."(1) RX) do. Res =. K * pyodd RX  end.
```

```
  if. y = {.″(1) Res
     do.
       S =. S, Res
NB.        wr 'K:',″(1)(″: K),″(1) ', R:',″(1) (″: RX),″(1) ' / ',″(1) (″:Res)
     end.
  i =. i + 1
end.
wr '*** Pythagorean by Leg ***'
~. }. S
)
```

付録　Ｊの正規表現プログラミングの要点（文献[1] より再録）

正規表現によるパターンの指定

・個々の文字	'abc', '3.14', 'にほんご'（２バイト文字も可能）	
・特殊文字	'¥.'⇔ピリオドそのもの, '¥('⇔かっこ	
・すべての文字	.　（ピリオド）	
・いずれか１文字	'[a-z]', '[A-Z]', '[0-9]'	
	'[aeiou]'	母音文字
	'[[:alpha:]]'	英文字
	'[[:alnum:]]'	英数文字
	'[[:digit:]]'	数字
	'[[:space:]]'	スペース
	'[[:punct:]]'	区切り文字
	'[[:cntrl:]]'	制御文字
・除いた文字	'[^aeiou]'	母音を除いた文字
・繰り返し * ０回～	'ab*c'	ac, abc, abbc, abbbc にマッチ
+ １回～	'ab+c'	abc, abbc, abbbc にマッチ
? ０回/１回	'ab?c'	ac, abc にマッチ
・文字列の繰り返し	かっこ( )でくくる	
・取り出し反復	かっこ( )でくくる	

正規表現の基本操作関数

・マッチした文字列にしるしをつける.

```
   '[[:digit:]]+' tryall 'abc234def43'
abc234def43
   ^^^   ^^
```

・マッチした文字列を取り出す.

```
   '[[:digit:]]+' rxall 'abc234def43'
+---+--+
|234|43|
+---+--+
```

・マッチした文字列を置換する.

```
   ('[[:digit:]]+';'***') rxrplc 'abc234def43'
abc***def***
```

・マッチした文字列に操作を行う.

```
  '[[:alpha:]]+' |. rxapply 'abc234def43'
cba234fed43
```