ベルギー数について

Belgian Numbers

慶応義塾大学理工学部
竹内寿一郎

１．はじめに

2000 年 12 月 16 日の JAPLA 研究会で山下紀幸氏が、既約分数を分子が 1 の分数の和で表現できる方式について、エジプトの分数式として紹介したことがあった。最近文献を漁っていて国の名前がついたものを見つけたのでここに紹介することにした。イギリス APL 協会発行の「ベクター」の中で、著者は高名な「ユージン・マクダネル氏」、そこに紹介されていたのは「ベルギー数」と名づけられていた数であった。最近の JAPLA 研究会でも鈴木 (義) 氏、西川氏、中村氏らが話題を撒いている「幸運数」など、ちょっとした整数論花盛りというところでしょうか。

今、ある数を考える。それを各桁に分け、繰返し並べた数のベクトルを作り、先頭から累積和ベクトルを計算し、その中に元の数が含まれる数を、ベルギー数と名づけるのである。

例えば 16 としよう。これからできるベクトルは、1 6 1 6 1 6 1 6 1 6 その累積和ベクトルは 1 7 8 14 15 21 22 28 29 35 で、これには 16 は含まれない。次に 17 の場合を計算してみよう。1 7 1 7 1 7 の累積和ベクトルは 1 8 9 16 17 25 でこれには 17 が含まれる。従って 16 はベルギー数ではないが、17 はベルギー数である。

そこで本稿ではこのベルギー数について検討することにしてみた。

２．J スクリプト

一桁の数は当然ベルギー数である。全て同一数からなる数はベルギー数である。二桁以上でも 1 と 0 からなる数字はベルギー数である。それらを拡張して一つ以上の同一数と一つ以上の 0 からなる数はベルギー数である。その他一般には簡単なルールが見つからない。そこで、ベルギー数を確かめるには手計算では大変なので、J スクリプトを作成して検討することにした。

まず、数値を一桁ずつ分解する。鈴木 (義) 氏による。

```
sep=:[:,[:".[:,.":
```

桁数によって必要な繰返し数を求める。

```
rep=:#@sep@]*[:>.(%+/@sep)"0
```

累積和ベクトルを計算する。

```
accum=:[:+/\(rep $ sep)
```

ベルギー数かどうか判定する。

```
jd=:[:*[:+/(=accum"0)
```

【jd の使い方の例】
●１から 100 までについて調べ、行列で表す。

```
   10 10$jd"0 >:i.100
```

●１～10000 まで 100 ずつ区切ってそれぞれ 100 個に含まれるベルギー数を数える。

```
   test=:3 : 0
qqq=:i.0[i=:_1
while. 9>:i=:>:i do.
qqq=:qqq,+/+/10 10 10$jd"0 [(1000*i)+>:i.1000
end.
)
```

qqq が 100 個中に存在するベルギー数の数からなるベクトルである。

３．結果

●まず、1～100 まで調べた結果を示す。

```
   10 10$jd"0 >:i.100
1 1 1 1 1 1 1 1 1 1
1 1 1 0 0 0 1 1 0 1
1 1 0 1 0 1 1 0 0 1
1 0 1 0 1 1 0 0 1 1
0 1 0 1 1 0 0 1 0 1
0 0 1 1 1 0 0 0 0 1
0 1 1 0 0 1 0 0 0 1
1 1 0 0 0 0 1 0 0 1
1 0 0 1 0 0 0 1 0 1
0 0 1 0 0 0 0 0 1 1
```

１１、１２，１３、１７、１８、２０がベルギー数であると分かる。
また、大きい数では８１、８４、８８、９０、９３、９９、１００がベルギー数。

●1～10000 までの各 100 個の中に存在するベルギー数の数。

```
   getexcel''
-1
   test''
100
   load 'plot'
   plot qqq
   (((#qqq),1)$qqq) dataout 1 1
```

ここから先、エクセルでグラフを描いた。

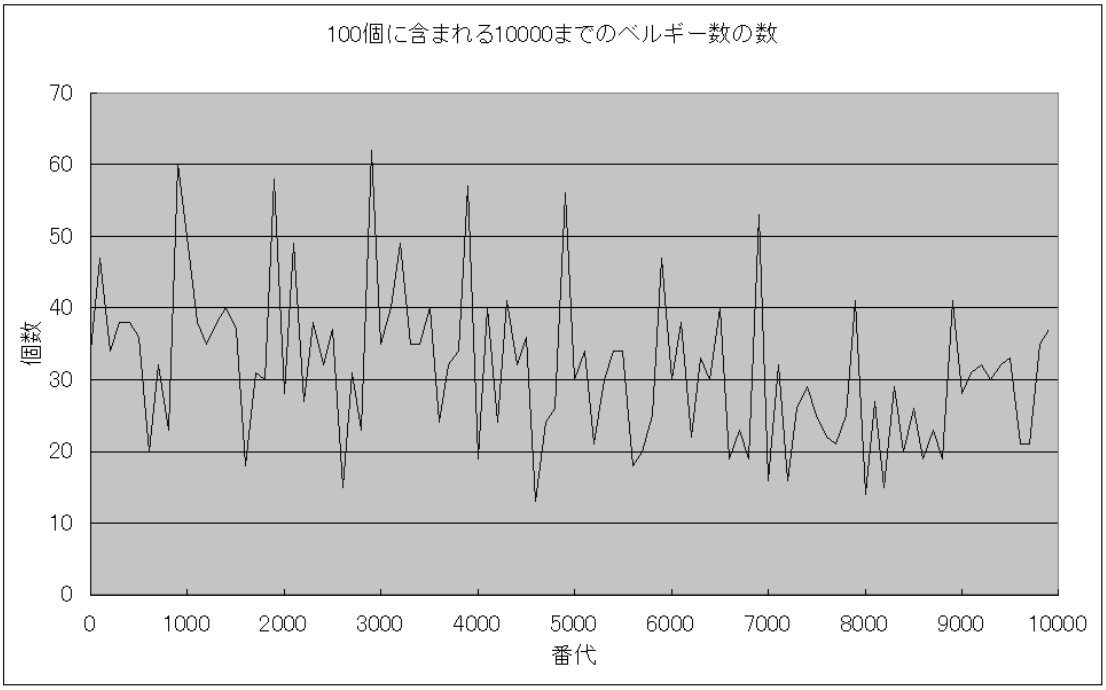

図１．1〜10000 までのベルギー数
100 個の中に存在する数

４．更なる検討

「ベクター」にも書かれているように、これまでの方法ではすぐにワークスペースがフルになってしまい、計算不可能で判定が出来なくなる。ベルギー数の見つけ方のうち、最も広く使わなければならない性質を考慮すべきであった。それは「各桁の和の数でその数が割り切れればベルギー数である」という性質である。つまり累積和の最大値が商の数だけ繰り返えされれば、もとの数になるからである。この性質を使えば大きな数でも、ワークスペースがフルにならずに判定できることが期待される。そこで以下にあるような int と jd2 関数を工夫してみた。

●大きな数だと Out of Memory になる。

```
   jd 123456789
0
   jd 1234567898
|out of memory: accum
|       jd 1234567898
```

まず、もとの数より小さい、桁数の和の整数倍のうち最大の数を求める。

```
   int=:3 : 'w*<.y.%w=.+/sep y.'
```

先に求めた最大数を基点にして累積和を求め、もとの数が含まれるかどうか判定する。

```
   jd2=:3 : 'y.e.www=:(0,+/\sep y.)+int y.
```

●この関数は小さな数でも判定できる。

```
   jd2"0 >:i.20
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 1
```

●大きな数を判定する。
表示桁数を増やす。

```
   9!:11(20)
   jd2 123456789
0
   jd2 1234567898
0
   jd2 1234567901
1
```

●ベクターに載っている例。

```
   jd2 1234567898765
1
```

このときの判定のための数値。

```
   5 3$www=:(0,+/\sep a)+int a=:1234567898765
1234567898744 1234567898745 1234567898747
1234567898750 1234567898754 1234567898759
1234567898765 1234567898772 1234567898780
1234567898789 1234567898797 1234567898804
1234567898810 1234567898815 1234567898744
```

数百桁もあるような数の判定は次の機会に譲ることにする。

【参考文献】

E.McDonnell(2005) : At Play with J:Belgian Numbers, VECTOR Vol.22 No.1 November 96-101.