$4x \pm 1$問題の研究

本田 真樹

平成 23 年 2 月 2 日

目 次

1　目的

与えられた数 x が偶数なら 2 で割り、奇数なら $3x+1$ にするという作業を繰り返すと、いつかは 1 になる。これをコラッツの問題といい、未解決である。コラッツの問題を解決することは厳しく、平成 19 年度卒業の平尾想太さんがコラッツの問題を変形した $3x-1$ の問題を研究しているので、今回は与えられた数 x が 3 の倍数なら 3 で割り、3 で割ったときの余りが 1 ならば $4x-1$、余りが 2 ならば類似した $4x+1$ にするという作業を繰り返す。するとどのようになるかについてを研究する。

2　方法

2.1　コラッツの問題

与えられた数 x が偶数なら 2 で割り、奇数なら $3x+1$ にするという作業を繰り返すと、いつかは 1 になるという予想である。これを関数で表すと

$$c(x) = \begin{cases} x/2 & (x \equiv 0 \bmod 2) \\ 3x+1 & (x \equiv 1 \bmod 2) \end{cases}$$

prolog を用いて実際に計算してみる。

```
/*コラッツの計算*/
collatz(N,M):- (N=:=2*(N//2)) -> M is N//2;M is 3*N+1.
collatz_d(N,M):- collatz(N,M),!.

start(1):- !.
start(N):- collatz_d(N,M),write(M),tab(2),!,
    start(M).
```

実行結果は次のようになる。

```
?- start(19).
58  29  88  44  22  11  34  17  52  26  13  40  20  10  5  16  8  4  2  1

?- start(78).
39  118  59  178  89  268  134  67  202  101  304  152  76  38  19  58  29
88  44  22  11  34  17  52  26  13  40  20  10  5  16  8  4  2  1
```

2.2　$3x-1$ の問題

2.2.1　$3x-1$ の計算

次に平尾さんが研究された問題について触れる。与えられた数 x が偶数であれば 2 で割り、奇数であれば $3x-1$ する。これを関数で表すと

$$h(x) = \begin{cases} x/2 & (x \equiv 0 \bmod 2) \\ 3x-1 & (x \equiv 1 \bmod 2) \end{cases}$$

コラッツの問題では必ず 1 になっているが、$3x-1$ の時には実際にどうなるのか、prolog で計算してみる。

```
/*3x-1 の計算*/
collatz2(N,M):- (N=:=2*(N//2)) -> M is N//2;M is 3*N-1.
collatz2_d(N,M):- collatz2(N,M),!.

as_j(J):- \+ j(J) -> asserta((j(J):- !)).

:- dynamic j/1.
start0(N):- collatz2_d(N,M),write(N),tab(2),
    as_j(M),asserta((j(M):-!)),start0(M).

start0_d(N):- start0(N).
start0_d(N).
```

実行結果は次のようになる。() 内はわかりやすいように足したもの。

```
?- start0_d(17).
17  50  25  74  37  110  55  164  82  41  122  61  182  91  272  136  68  34  17

Yes
?- start0_d(53).
53  158  79  236  118  59  176  88  44  22  11  32  16  8  4  2  1

Yes
?- start0_d(19).
19  56  28  14  7  20  10  5

Yes
?- start0_d(78).
78  39  116  58  29  86  43  128  64  (32  16  8  4  2  1)

Yes
```

2.2.2　結果

この結果から、平尾さんは終わり方には次の 3 パターンの場合があるとし、これを予想している。

予想される 3 つの種類

1. いつかは 1 になる

   例： $15, 44, 22, 11, 32, 16, 8, 4, 2, 1$

2. 循環する

   例： $5, 14, 7, 20, 10, 5$

   結局循環のパターンは 2 つであると予想される。

   (1). $5, 14, 7, 20, 10, 5$

   (2). $17, 50, 25, 74, 37, 110, 55, 164, 82, 41, 122, 61, 182, 91, 272, 136, 68, 34, 17$

3. 途中から循環する

   例：$36, 18, 9, 26, 13, 38, 19, 56, 28, 14, 7, 20, 10, 5, 14$

   これは、上で述べた 2 つの循環節に途中からはいるというものである。

2.3 $4x \pm 1$ の問題

2.3.1 $4x \pm 1$ の計算

与えられた数 x が 3 の倍数なら 3 で割り、3 で割ったときの余りが 1 ならば $4x-1$、余りが 2 ならば $4x+1$ をするという作業を繰り返す。これを関数で表すと

$$f(x) = \begin{cases} 4x-1 & (x \equiv 1 \bmod 3) \\ 4x+1 & (x \equiv 2 \bmod 3) \\ x/3 & (x \equiv 0 \bmod 3) \end{cases}$$

実際にどのようになるか prolog で計算してみる。

```
/*4x ± 1 の計算*/
tri(N,Q,R) :- Q is N//3,R is N - 3*Q.

honda(N,M) :- N>1,
    tri(N,Q,R),!,
    ((R==0) -> M is N//3;
    ((R==1) -> M is 4*N -1);
    M is 4*N +1).

as_h(J):- \+ h(J) -> asserta((h(J):- !)).

:- dynamic h/1.
masaki(N,C) :- C>0,
    honda(N,M),
    write([C]),put(9),write(M),put(9),!,as_h(M),
    asserta((h(M):-!)),
    C1 is C + 1,
    masaki(M,C1).

masaki_h(N,C):- masaki(N,C).
masaki_h(N,C).
```

実行結果は次のようになる。

```
?- masaki_h(5,1).
[1]     21      [2]     7       [3]     27      [4]     9       [5]     3
[6]     1
Yes
```

```
?- masaki_h(19,1).
[1]     75      [2]     25      [3]     99      [4]     33      [5]     11
[6]     45      [7]     15      [8]     5       [9]     21      [10]    7
[11]    27      [12]    9       [13]    3       [14]    1
Yes

?- masaki_h(78,1).
[1]     26      [2]     105     [3]     35      [4]     141     [5]     47
[6]     189     [7]     63      [8]     21      [9]     7       [10]    27
[11]    9       [12]    3       [13]    1
Yes
```

2.3.2 $4x \pm 1$ の計算 2

ここでこの計算を与えた範囲で行うプログラムを作る。

```
/*4x ± 1 の計算 2*/
for(I=<J,I):- I=<J.
for(I=<J,K):- I=<J,
    I1 is I+1,for(I1=<J,K).

masaki1(N,K,C):- for(N=<K,I),nl,write(I),nl,masaki(I,C),nl.

masaki2(N,K,C):- masaki1(N,K,C).
masaki2(N,K,C).
```

実行結果は次のようになる。

```
?- masaki2(5,100,1).
5
[1]     21      [2]     7       [3]     27      [4]     9       [5]     3
[6]     1
6
[1]     2       [2]     9       5 の列に入る
7
[1]     27                      5 の列に入る
8
[1]     33      [2]     11      [3]     45      [4]     15      [5]     5
[6]     21                      5 の列に入る
9
[1]     3                       5 の列に入る
```

10

[1]	39	[2]	13	[3]	51	[4]	17	[5]	6
[6]	23	[7]	93	[8]	31	[9]	123	[10]	41
[11]	165	[12]	55	[13]	219	[14]	73	[15]	291
[16]	97	[17]	387	[18]	129	[19]	43	[20]	171
[21]	57	[22]	19	[23]	75	[24]	25	[25]	99
[26]	33			8 の列に入る					

11

[1]	45			8 の列に入る					

12

[1]	4	[2]	15	8 の列に入る					

13

[1]	51			10 の列に入る					

14

[1]	57			10 の列に入る					

15

[1]	5			5 の列に入る					

16

[1]	63	[2]	21	5 の列に入る					

17

[1]	69			10 の列に入る					

18

[1]	6	[2]	2	6 の列に入る					

19

[1]	75			10 の列に入る					

20

[1]	81	[2]	27	5 の列に入る					

21

[1]	7			5 の列に入る					

22

[1]	87	[2]	29	[3]	117	[4]	39		
				10 の列に入る					

23

[1]	93			10 の列に入る					

24

[1]	8	[2]	33	8 の列に入る					

25

[1]	99			10 の列に入る					

26
[1] 105 [2] 35 [3] 141 [4] 47 [5] 189
[6] 63 16 の列に入る
27
[1] 9 5 の列に入る
28
[1] 111 [2] 37 [3] 147 [4] 49 [5] 195
[6] 65 [7] 261 [8] 87
22 の列に入る
29
[1] 117 10 の列に入る
30
[1] 10 [2] 39 10 の列に入る
31
[1] 123 10 の列に入る
32
[1] 129 10 の列に入る
33
[1] 11 8 の列に入る
34
[1] 135 [2] 45 8 の列に入る
35
[1] 141 26 の列に入る
36
[1] 12 [2] 4 12 の列に入る
37
[1] 147 28 の列に入る
38
[1] 153 [2] 51 10 の列に入る
39
[1] 13 10 の列に入る
40
[1] 159 [2] 53 [3] 213 [4] 71 [5] 285
[6] 95 [7] 381 [8] 127 [9] 507 [10] 169
[11] 675 [12] 225 [13] 75
10 の列に入る
41
[1] 165 10 の列に入る

42
[1] 14 [2] 57 10 の列に入る
43
[1] 171 10 の列に入る
44
[1] 177 [2] 59 [3] 237 [4] 79 [5] 315
[6] 105 26 の列に入る
45
[1] 15 8 の列に入る
46
[1] 183 [2] 61 [3] 243 [4] 81
20 の列に入る
47
[1] 189 26 の列に入る
48
[1] 16 [2] 63 16 の列に入る
49
[1] 195 28 の列に入る
50
[1] 201 [2] 67 [3] 267 [4] 89 [5] 357
[6] 119 [7] 477 [8] 159
40 の列に入る
・
・
・

3　結果

これらの計算結果から、自然数 n からスタートすればいつかは 1 になると予想される。そこで n からスタートし、この計算をして 1 になることを予想 $H(n)$ とおく。そして $H(n)$ の成立しない最小の数を n とする。n からスタートして、n より小さい数 $H(m)$ は成立するので、$H(n)$ も成立する。よってこのプログラムでは与えられた数 n より小さい数が出ればそれは 1 になると判断出来る。

3.1　予想されるパターン

3.1.1　3 の倍数の場合

与えられた数 n が 3 の倍数の場合、最初の計算をすることで与えられた数 n より小さくなるのでそこで計算は終了する。

3.1.2　3 の倍数でない場合

振り子のように大きくなることと、小さくなることを繰り返しながら最終的には 1 になる。この時早い段階で小さくなることもあるが、一度大きくなる場合もある。

4　考察

4.1　予想の検証

前章での予想が実際に成り立つのか 100000 まで検証してみる。また 100000 で行った計算をグラフにしてみる。

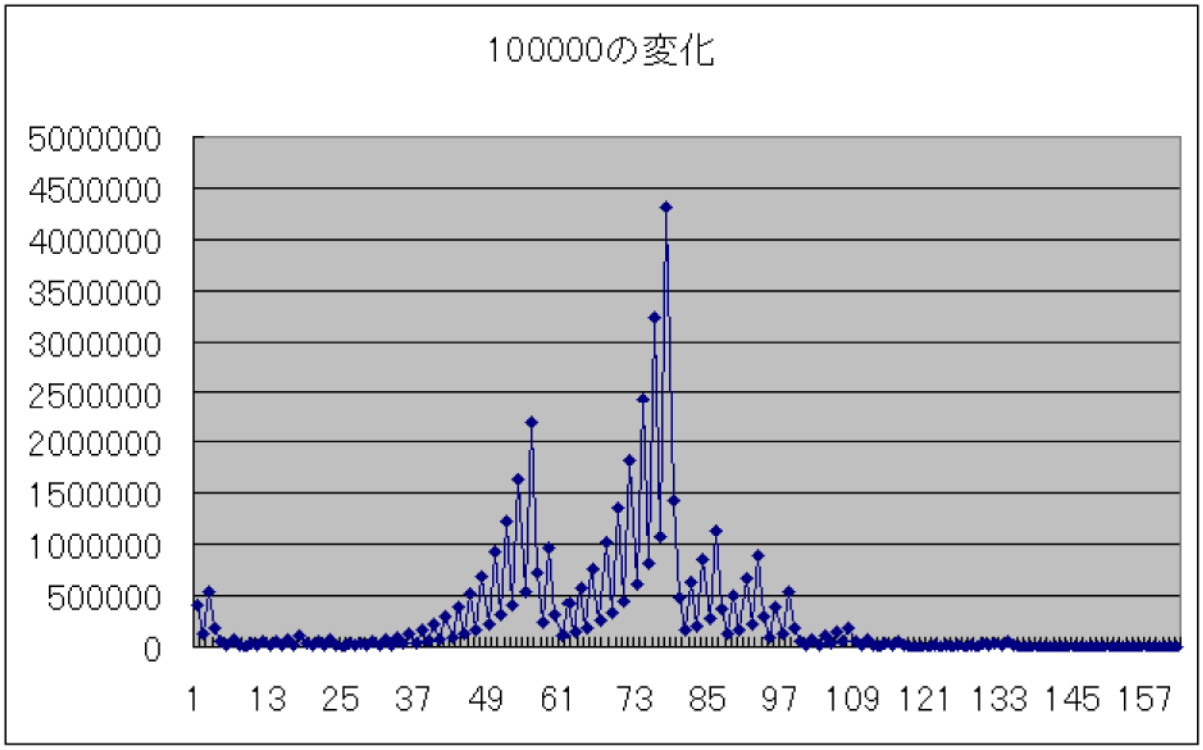


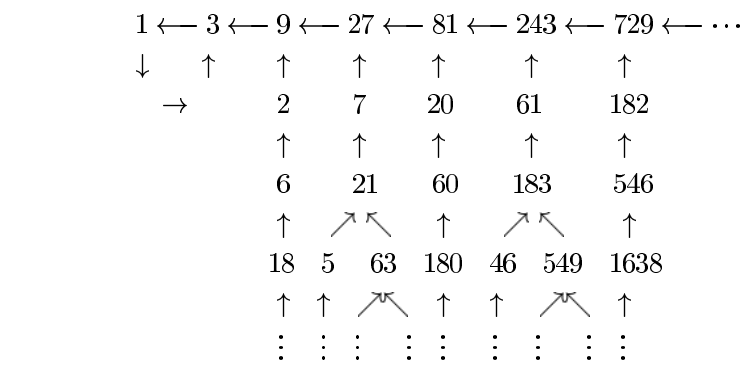
図 1: 100000 の変化

すると大きくなるのと、小さくなるのを繰り返しながら段々振り幅が大きくなり、1 番大きい数で 4315653 まで跳ね上がる。しかしそこから段々と振り幅が小さくなり始め最終的に 1 になる。1 になるまでに 162 回もの計算がされていた。

4.2　逆向きの演算

今までやってきた演算を逆から行う。3^n は必ず 1 にいくことは明らかなので、その数列をもとに考えてみる。

1　3　9　27　81　243　729$\cdots$



ここで注目するのは 3 の倍数と 6 の倍数である。まず 3 の倍数で 9 を見てみると、27 を 3 で割っても 9 だが、2 を 4 倍して 1 を足しても 9 である。つまり 3 の倍数で分岐する。しかし 1 つ例外があり、3 の倍数であっても偶数でもある 6 の倍数は分岐せずに 3 倍されていくことがわかった。
これを関数でまとめる。$f(x) = y$ とおくと、上の条件から $y \equiv 3 \bmod 6$ が成り立つものは 2 つに分岐する。

4.3　逆向きの演算のプログラム

基本的には 3 倍していくが、出てくる数の中で 1 を引くか足すかをし、4 で割り切れるものは 2 つに分岐する。このプログラムでは 3 倍でない方の分岐を優先し、1000 を越えないところまでを出力する。

```
third(K,J):- K=:=3*(K//3),
        ((0=:=(K-1) mod 4) -> J is (K-1)//4;
        (0=:=(K+1) mod 4) -> J is (K+1)//4).
third(K,J):- J is K*3.

as_h(J):- \+ h(J) -> asserta((h(J):- !)).

:- dynamic h/1.
pon(K):- third(K,J),write(K),put(9),!,as_h(K),
        asserta((h(K):-!)),
        ((J>=1000)-> fail;pon(J)).

for(I=<J,I):- I=<J.
for(I=<J,K):- I=<J,
    I1 is I+1,for(I1=<J,K).

pon1(K,L):- for(K=<L,I),nl,pon(I),nl.

ponzu(K,L):- pon1(K,L).
ponzu(K,L).
```

すると

```
?- ponzu(1,100).

1       3       1
2       6       18      54      162     486
3
4       12      36      108     324     972
5       15      4
6
7       21      5
8       24      72      216     648
9       2
10      30      90      270     810
```

11	33	8	
12			
13	39	10	
14	42	126	378
15			
16	48	144	432
17	51	13	
18			
19	57	14	
20	60	180	540
21			
22	66	198	594
23	69	17	
24			
25	75	19	
26	78	234	702
27	7		
28	84	252	756
29	87	22	
30			
31	93	23	
32	96	288	864
33			
34	102	306	918
35	105	26	
36			
37	111	28	
38	114	342	
39			
40	120	360	
41	123	31	
42			
43	129	32	
44	132	396	
45	11		
46	138	414	
47	141	35	
48			
49	147	37	

50　　150　　450
・
・
・

4.4　逆向きの演算のまとめ

予想したように 6 の倍数はそのまま 3 倍されていき、$y \equiv 3 \bmod 6$ が成り立つものは分岐した方の数が出ている。数値の範囲を広げて検証してみたが、同様の結果になった。また出てきた数を並び替えてみると、自然数を網羅する。よって逆向きの演算からも、$4x \pm 1$ の計算において自然数は 1 になることがわかった。

4.5 $5x+1$ の計算

今までの計算は全て循環をしている。1 になるコラッツの計算や $4x \pm 1$ の計算も 1 を最終的なものとして見ているが、$4x \pm 1$ の逆演算で示した通りに 1 と 3 で循環している。そこで同様の計算で循環しない場合があるものがないか試してみる。コラッツの計算を更に変更し、与えられた数 x が偶数ならば 2 で割り、奇数ならば $5x+1$ をする作業を繰り返すとどのようになるのか prolog を用いて計算してみた。すると循環もするが、与えた数によっては無限に大きくなる場合もあるようだ。他にも様々なパターンを試してみたが、必ず循環するパターンは見つからなかった。よって全てのパターンで循環するわけではなく、コラッツの問題や $3x-1$、$4x \pm 1$ などの循環のみをするパターンは限られているのだと思われる。

参考文献

[1] 平尾 想太　平成 19 年度卒業論文「3x-1 の問題」

5　感想

最初にいくつかの適当な数字で実際に計算をしてみて、それが全部 1 になったときは何かしら例外があるのではと思いました。しかし prolog を用いて計算をしてみたら、自分では相当な時間がかかるであろう大きな数も 1 になることがわかりました。しかもそれが自分の作ったプログラムということもありとても充実感がありました。自分はこのゼミに入るまで飯高先生とは一切関わりがありませんでしたが、大学生活最後の一年間を先生に見てもらいとても楽しかったです。本当にありがとうございました。先生もお体に気をつけて下さい。