

## ① 予備知識

漸化式

$$F_0 = 0, \quad F_1 = 1, \quad F_{n+2} = F_{n+1} + F_n \quad (n \geq 0)$$

で定義される数列  $\{F_n\}$  を Fibonacci 数列、Fibonacci 数列に現れる数を Fibonacci 数と呼びます。

$n$	0	1	2	3	4	5	6	7	8	9	10	11	...
$F_n$	0	1	1	2	3	5	8	13	21	34	55	89	...

②  $n$  番目の Fibonacci 数

Maxima を使って、 $n$  番目の Fibonacci 数を求める関数を作ってみたいと思います<sup>\*1</sup>。

```
(%i1) fib_r(n) := block(
  if n = 0 or n = 1 then
    return(n)
  else
    return(fib_r(n - 1) + fib_r(n - 2))
);
(%o1) fib_r(n) := block(if (n = 0) or (n = 1) then return(n)
  else return(fib_r(n - 1) + fib_r(n - 2)))
(%i2) fib_r(6);
(%o2) 8
```

関数 `fib_r` は再起呼び出しを用いた素朴な方法です<sup>\*2</sup>。

## ③ Fibonacci 数列の行列表現

Fibonacci 数列の定義式により、任意の自然数  $n$  に対して、

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}$$

が成り立ちます。従って、

$$\begin{aligned} \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^2 \begin{pmatrix} F_{n-1} & F_{n-2} \\ F_{n-2} & F_{n-3} \end{pmatrix} = \dots = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \end{aligned}$$

<sup>\*1</sup> Maxima には Fibonacci 数を求める関数 `fib` が用意されています。

<sup>\*2</sup> 関数 `fib_r` の引数は、非負整数でなければなりません。

と表せることが分かります<sup>\*3</sup>。この式から、 $n$  より小さい自然数  $d$  に対して、

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^d \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-d} \quad \text{より} \quad \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} F_{d+1} & F_d \\ F_d & F_{d-1} \end{pmatrix} \begin{pmatrix} F_{n+1-d} & F_{n-d} \\ F_{n-d} & F_{n-1-d} \end{pmatrix}$$

が成り立つことが分かりますが、特に、(2, 1) 成分を比較することにより、関係式

$$F_n = F_d F_{n+1-d} + F_{d-1} F_{n-d} \quad (\spadesuit)$$

が得られます。関係式 ( $\spadesuit$ ) において、 $n$  が偶数  $2k$  の場合、 $d = \frac{n}{2} = k$  ととると、

$$\begin{aligned} F_n &= F_k F_{2k+1-k} + F_{k-1} F_{2k-k} = F_k F_{k+1} + F_{k-1} F_k = F_k (F_k + F_{k-1}) + F_{k-1} F_k \\ &= F_k (F_k + 2F_{k-1}) \end{aligned} \quad [1]$$

が得られます。一方、奇数  $2k-1$  の場合は、 $d = \frac{n+1}{2} = k$  ととると、

$$F_n = F_k F_{(2k-1)+1-k} + F_{k-1} F_{(2k-1)-k} = F_k^2 + F_{k-1}^2 \quad [2]$$

が得られます。

#### ④ 改良

第2節で作成した関数 `fib_r` は、非常に効率の悪いプログラムですが、前節の結果を用いれば、改良できるかもしれません。

```

1  fib_s(n) := block([k],                                     -fib_s.mac-
2      if n = 0 or n = 1 then
3          return(n)
4      elseif evenp(n) then (
5          k: n/2,
6          return(fib_2(k) * (fib_2(k) + 2 * fib_2(k-1)))
7      ) else (
8          k: (n + 1) / 2,
9          return(fib_2(k)^2 + fib_2(k-1)^2)
10     )
11 );

```

5行目と6行目が偶数の場合の公式 [1] に相当し、8行目と9行目が奇数の場合の公式 [2] です。

改良版 `fib_s` と素朴な方法 `fib_r` の2通りで、30番目の Fibonacci 数を計算したところ、次のような結果となりました<sup>\*4</sup>。

<sup>\*3</sup> 厳密には、仮定  $n \geq 2$  のもとで数学的帰納法で証明し、更に、 $n = 1$  のときにも成り立つことを確認すべきです。

<sup>\*4</sup> 実験に使った CPU は、Core 2 Duo 2.66GHz です。

```
(%i3) load("fib_s.mac");
(%o3) fib_s.mac
(%i4) showtime: ture;
Evaluation took 0.0000 seconds (0.0000 elapsed) using 56 bytes.
(%o4) ture
(%i5) fib_s(30);
Evaluation took 0.0000 seconds (0.0000 elapsed) using 87.813 KB.
(%o5) 832040
(%i6) fib_r(30);
Evaluation took 44.6900 seconds (44.8000 elapsed) using 1530.773 MB.
(%o6) 832040
```

素朴な方法では45秒近くかかったのに対して、改良版ではほとんど時間はかからなかったことから、ちゃんと改良できているようです。

## ⑤ 念のため補足

再帰呼び出しにこだわらず、初期値から順番に求める手順の方が高速なはずです。

```

1  fib_t(n) := block([c, a: 0, b: 1],
2  for i: 1 thru n - 1 do (
3  c: a + b,
4  a: b,
5  b: c
6  ),
7  return(c)
8  );

```

- fib\_t.mac -

また、誤差評価が必要になりますが、一般項  $F_n = \frac{\phi^n - (1 - \phi)^n}{\sqrt{5}}$  を用いる方法もあります。

ここで、 $\phi$  は黄金比  $\frac{1 + \sqrt{5}}{2}$  です。