3.2 マチンの計略

円周率の値を計算するのに無限級数を用いた。ただ、そこで用いた級数は収束が遅いため計算機向きでなかった。計算機向きの級数に仕立て上げたのはマチン[1]である。詳しい経過を述べることはできないが、マチンは

$$\frac{\pi}{4} = 4\left(\frac{1}{1\cdot 5} - \frac{1}{3\cdot 5^3} + \frac{1}{5\cdot 5^5} - \cdots\right) - \left(\frac{1}{1\cdot 239} - \frac{1}{3\cdot 239^3} + \frac{1}{5\cdot 239^5} - \cdots\right)$$

という式をひねり出した。ここでもまた $\dfrac{\pi}{4} = \cdots$ の形である。気になる人にだけ、そっと耳打ちしておこう。グレゴリーの式もマチンの式も $\tan\dfrac{\pi}{4} = 1$ であること、すなわち $\dfrac{\pi}{4} = \arctan 1$ であることが利用されているからだ。そして

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots$$

である。以上。え？　なおさら気になる？　知りたい人はきちんとした数学の書物を読んでみよう。

さて、いまはマチンの式を

$$\frac{\pi}{4} = \left(\frac{4}{1\cdot 5} - \frac{1}{1\cdot 239}\right) - \left(\frac{4}{3\cdot 5^3} - \frac{1}{3\cdot 239^3}\right) + \left(\frac{4}{5\cdot 5^5} - \frac{1}{5\cdot 239^5}\right) - \cdots$$

と見て、π の値を計算することにしよう。

[py script]

```
>>> def fx(n):
...   u = 1; v = 1
...   for n in range(1, n+1):
...     u *= 5
...     v *= 239
...   return (4 / (n * u) - 1 / (n * v))
...
>>> def mpi(n):
...   p = 0
...   sgn = 1
...   for n in range(1, n, 2):
...     p += sgn * fx(n)
...     sgn = -sgn
...   print(4 * p)
...
>>> mpi(20)
3.1415926535897927
```

スクリプトはだいぶ無駄が多く、少々不細工かもしれないが、それでもわずかの項数計算でそこそこの値が出たことを素直に喜ぼう。ただ、スクリプトをざっと見ると、関数がふたつあることに

[1] ジョン・マチン（1685–1751）：イギリスの天文学者。

気づくはずだ。ひとつの関数にまとめて書くこともできるが、今後のためにも、そして将来のためにも複数の関数を扱えるようにしよう。

マチンの式は複雑であるが、実際は一種類の項を、繰り返し足したり引いたりしているだけである。基本となる第 n 項は

$$\frac{4}{(2n-1)5^{2n-1}} - \frac{1}{(2n-1)239^{2n-1}}$$

という式であり、これらを交互に足したり引いたりしている。ここの処理さえ上手にできれば、あとはいま来た道をたどるだけでよい。ここは分母が単なる奇数である分数と違い、n の値によって分母が変わってくる。ならば、n の値によって正しい値を返す関数を定義すればよいだろう。

その面倒な計算は関数 `fx(n)` が請け負う。味気ない名称は許してほしい。`fx` 関数は項番号を整数値 `n` で受け取り、面倒な $\frac{4}{(2n-1)5^{2n-1}} - \frac{1}{(2n-1)239^{2n-1}}$ を計算したあと、`return` 文で値を返す。ここで、`fx` 関数内部の `for` 文の繰り返しが `n+1` までになっていることに注意してほしい。何度も言うようだが、`range` 関数の第 2 引数に `n` を与えても `n-1` までの繰り返ししかしないからだ。その上で、べき乗の計算は指数関数を使わず `for` 構文で処理している。見てのとおり `n` が 3 を受け取れば 5^3 と 239^3 が、`n` が 7 を受け取れば 5^7 と 239^7 が計算される仕組みだ。以上で面倒な式計算の結果が返されるのだ。

ちなみに、`u = 1; v = 1` と書いた部分がある。本来なら、1 行に式ひとつが原則である。どうしても行数をケチりたいときは、“;” で列挙できるという見本であるが、あまり多用しないほうがよいだろう。もし行をケチるなら、いっそ `u = v = 1` としてしまおう。

これで $\frac{4}{(2n-1)5^{2n-1}} - \frac{1}{(2n-1)239^{2n-1}}$ を計算する `fx` 関数が定義できた。enter キーを押してプロンプトを “`>>> `” にして、肝心の $\frac{\pi}{4}$ を求める関数を定義しよう。といっても、前の関数と同じである。`p += sgn / n` が `p += sgn * fx(n)` に変わっただけだ。つまり `mpi` 関数は、計算の一部を下請けに—関数 `fx(n)` に—委ねたということである[2]。

このように、計算式が複雑になりそうなら、そういう部分を関数に分けてしまうのが効率的だ。そうすれば、面倒な計算の処理に悩むことなく、本来の処理に集中できるからだ。対話型プログラミングで関数をふたつ以上定義する場合は、定義する順番はどうでもよい。いまの場合なら、`mpi` 関数から先に入力してもよい。そして、一度入力した関数は、**Python3** を終了させるまで利用できる。

さて、実際の使い方は `mpi(20)` のようにする。これで、`n = 20` として `mpi` 関数が処理され、同時に `fx` 関数は `fx(20)` で計算される。`mpi` 関数が `fx` 関数を呼び出すのだ。`n = 20` で計算すると

[2] version 2.x の場合は、`return` 文の数値に小数点が必要である。

いうことは、マチンの式を 10 項だけ計算することである。末尾は四捨五入されているのだけれど、この値は小数点以下 14 桁まで正しい。このことからもマチンの式の収束の速さが分かるというものだ。

実は、10 項の計算で十分なのである。**Python3** は小数点以下 16 桁まで表示しているけれど、これ以上誤差をなくせないので 5^k や 239^k が小数点以下 16 桁を下回る項まで計算するのは無駄だ。どこの項まで計算させるかの調べ方にはいろいろあるが、この先の散歩道で目にするはずなので、ちょっと待ってもらいたい。