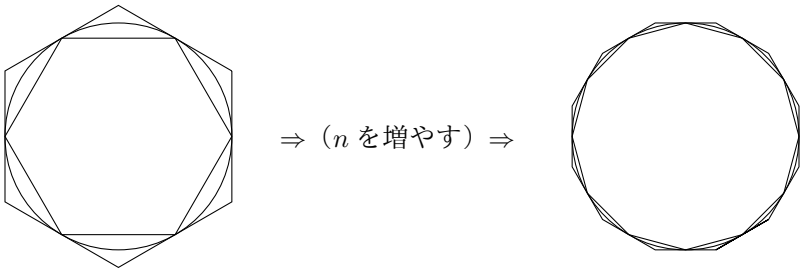
3. . . の夢見鳥

3.1 円周率

円周率は不思議なもので、古くから多くの人々を魅了してきた。いちばん簡単に円周率を表す数は 3 である。また、もっとも効率のよい近似値なら 3.14 というところだろうか。円周率は、小数点以下に不規則な数が並ぶので、昔から有効桁数を高める競争が行われてきた。コンピュータが発達した現代でも、それは続けられている。

円周率を求める古典的な方法は、アルキメデス[1)]が考案した。円に内接する正 n 角形の周長と円に外接する正 n 角形の周長を計算し、それらにはさまれた値を円周率とすることである。はさまれた値といっても、ある範囲にはさまれてるわけだから、値が確定しない。したがって、内接 n 角形の周長と外接 n 角形の周長で、一致している部分までが円周率の正しい値を示していることになる。



これらの計算をするには、三角比に関する知識があるとよいのだが、ここではアルキメデスの方法で円周率の近似をすることが目的ではない。期待した諸君には申し訳ないが省略させてもらおう。

円周率はギリシア文字 π で代用される。円周率が通常の分数で表せないので当然の処置だろう。数値計算をするソフトウェアには大抵、円周率の値が組み込まれていて、いつでも自由に呼び出せるものだが、TEX は数値計算をするためのソフトウェアではないから当然そんな値は保持していない。でも Lua にはあるので、もちろん LuaTEX は円周率が使える。『`\directlua{tex.print{math.pi}}`』と書いて『3.1415926535898』が出力される。

1) シラクサのアルキメデス（287?B.C.–212B.C.）：古代ギリシアの数学・物理学者。

組版の最中に単に円周率の値が必要になったら、この値で十分お釣りがくる。でも、われわれは実用なんて言葉とは無縁である。ただ円周率の値、というより数字の羅列が見たいのだ。夢見鳥が翔べる範囲にそんな景色はあるだろうか。

円周率は通常の分数で表せないけれど

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots$$

といった、**無限級数**で表すことは可能だ。これはグレゴリー[2)]もしくはライプニッツ[3)]の功績だ。式が $\pi = \ldots$ ではなく $\frac{\pi}{4} = \ldots$ と書いてあるのは、その求め方に由来しているが、詳しいことは別の書物を参考にしてもらいたい。右辺を無限に計算することができるなら、その結果はぴったり円周率の値となる。しかし、これは計算機向きの式ではない。なぜなら**収束**が非常に遅いからだ。収束が遅いことは、計算機の処理速度がいかに速くても致命的なものである。

収束が遅いことは式を見ているだけでも理解できると思う。たとえば級数を 5 億項先まで加えても、分母は 10 億程度の大きさである。これでは小数点以下 8 桁の精度にしかならない。

しかし悲観ばかりしていても仕方ない。収束が遅いことを承知の上で、この方法を LuaTEX で再現しよう。そこで、先の無限級数の $\frac{1}{n}$ の項までの和をとる関数の定義からだ。それを『

```
\begin{luacode*}
function qpi(n)
    p = 0  sgn = 1
    for i = 1, n, 2 do
        p = p + sgn / i
        sgn = -sgn
    end
    return 4 * p
end
\end{luacode*}
```

』としてみた。これで『`\directlua{tex.print(qpi(201))}`』と書けば『3.151493401071』が出力される。スクリプト名は π の四分の一（quater of pi）を求める計算式にちなんだ。

スクリプトにはいくつかの見所がある。`p` は $\frac{\pi}{4}$ の値を格納する変数であることはよいとして、`sgn` ってなんだろう。これは符号の代わりである。級数は交互に足したり引いたりしているため、

2） ジェームス・グレゴリー（1638–1675）：イギリスの数学者・発明家。
3） ゴットフリード・ウィルヘルム・ライプニッツ（1646–1716）：ドイツの哲学・数学者。

足す項には +、引く項には - を与えたいのである。そうすれば `p = p + sgn / i` の計算が足す・引く交互にできる。

もちろん奇数項と偶数項で足す・引くを変えるなら、たとえば `if` 文を用いて

```
for i = 1, n do
    if i % 2 == 0 then
        p = p - 1 / (2 * i - 1)
    else
        p = p + 1 / (2 * i - 1)
    end
end
```

と書いてもよいだろう。この方がもとの式に忠実で分かりやすい。なぜならもとの式は一般項を書き加えると

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots \begin{cases} -\dfrac{1}{2n-1} & (n\text{:偶数}) \\ +\dfrac{1}{2n-1} & (n\text{:奇数}) \end{cases} + \cdots$$

だからだ。`if` 文を用いたスクリプトの場合、`n` は第 n 項の意味になる。したがって、`for` 文を差し替えたスクリプトで `\qpi(201)` と書いたならそれは 201 項目、すなわち $\dfrac{1}{2\times 201-1} = \dfrac{1}{401}$ までの和である。

もともとのスクリプトが `p = p + 1 / (2 * i - 1)` のようになってないことに注意してほしい。それは `for i = 1, n, 2` と書いたからである。`for` 文で 3 番目に与える数は `i` の増分を意味する。よって、この書き方で `i` には 1, 3, 5, 7, ... が代入されるので、直接 `p = p + sgn / i` と書けたのだ。

大事なことを一つ言おう。どちらが優れたスクリプトだろうか、と問うことに意味はない。もともとのスクリプトは「$\dfrac{1}{n}$ までの和」を求めるスクリプトで、`if` 文に差し替えたスクリプトは「第 n 項までの和」を求めるスクリプトになってるにすぎない。要は `qpi(100)` と書いたとき、それが $\dfrac{1}{100}$ までの和なのか、第 100 項までの和なのかを把握していることが重要なのだ。つまり `qpi(n)` の**仕様**を知っていることである。

ちなみに、もとのスクリプトで `qpi(100)` を処理するのはなんの問題もない。`for` 文は `99` まで処理したら何食わぬ顔でループを抜ける。

結果を見て分かる通り、`qpi(201)` で $\dfrac{1}{201}$ までの和を計算をした割には、よく知られた値 3.14 とは 0.01 も違う。『`tex.print(qpi(50000000))`』の出力でようやく『3.1415926135898』となった。

しかも、出力までに少々時間がかかっていながら小数点以下 7 桁目までしか正しくない。それは当然のことで、$\frac{1}{50000000}$ の精度は小数点以下 8 桁である。LuaTEX の 12 桁の精度とは比べるべくもない。夢見鳥の飛翔路も楽じゃない。