## 2.3 大きな数の扱い方

指数関数の計算をしてみると分かることだが、計算結果は爆発的に大きくなってしまう。一般にコンピュータでは、ひとつの変数が扱える整数は場合によっては数万程度の大きさしかないから、すぐに桁あふれを起こしてしまうのだ。$y = 2^x$ といった単純な関数でさえ、コンピュータはついて行けない。$x = 30$ でもコンピュータは悲鳴をあげるだろう。$x = 80$ ともなればお手上げだ。ただし、**Excel** のワークシートでは $2^{1000}$ であってもかんしゃくをおこさず値を求めてくれる。しかし、何度も言うように概数でだ。

**Excel** で大きな数を正確に扱うことは不可能なのだろうか？　もちろんそんなことはない。解決のヒントは、われわれが何気なく使っている数に隠れている。われわれは機械と違って、数を記憶するための制限を持たない。だから、10 億でも 1 兆でも扱える。本当にそう思っている？　実はわれわれのほうがコンピュータより貧弱だ、と言ったらどう考えるだろうか。だが、これは事実なのである。コンピュータはひとつの変数で数万程度の大きさしか扱えないと言ったが、われわれはひとつの桁で 9 までの数しか扱えないことを知っているだろうか。

そろそろ私の言いたいことが見えてきたことと思う。つまり、こういうことだ。われわれは 0 〜 9 までの 10 種類—今後のためにあえて言うが、1 〜 0 までの 10 種類ではない—の数しか扱えないが、位取りをしているために望むだけ大きな数を扱えるのである。これをコンピュータにも応用しよう。ひとつの変数で数万の数しか扱えなくても、位取りの考えを持ち込むことで大きな数を扱うことができるのである。普通コンピュータは 2 進数で演算していると言われるが、ひとつの変数をひとつの桁にすれば数万進数で演算することも可能なのだ。われわれが貧弱と言ったのは、そういう意味である。

では、指数関数の計算に入る前に**配列**について話しておこう。

programming list [Arrays.vba]

```
 1 : Sub Arrays()
 2 : Dim f(3), fn, rn As Integer
 3 :
 4 :     f(0) = 2: f(1) = 5: f(2) = 6
 5 :
 6 :     fn = 100 * f(0) + 10 * f(1) + f(2)
 7 :     rn = 100 * f(2) + 10 * f(1) + f(0)
 8 :
 9 :     Sheet1.Cells(2, 1) = fn
10 :     Sheet1.Cells(3, 1) = rn
11 : End Sub
```

配列による変数を宣言する場合は、() 付きの変数名を用いる。2: 行目の `f(3)` がそうだ。`f(3)` と書いたら 3 つの変数 `f` が、`g(10)` と書いたら 10 個の変数 `g` が用意されるのだが、注意しなくて

はならないことがある。それは、f(3) で用意される 3 つの変数は f(0), f(1), f(2) の 3 つであり、g(10) で用意される 10 個の変数は g(0), g(1), . . . , g(9) の 10 個である点だ。つまり f(n) という配列は、f(0) から f(n - 1) までの $n$ 個の変数になるのである。このことは、$n$ 進法で 0 から $n-1$ までの $n$ 個の数が使われることと合致している。さっき 0 〜 9 までの 10 種類の数を扱うと強調したのは、ここでの説明を見越してのことだ。そういうわけで、今は f という名の配列を宣言し、f(0), f(1), f(2) の 3 変数が準備されたことになる。

さて、マクロは 3 桁の整数を模している。2: 行目の Dim f(3) によって 3 桁分の "位" を用意した。位というのは少し変だが、この場合の処理ではそうしている。つまり f(0) の位、f(1) の位、f(2) の位が用意されたことになる。

4: 行目で位に使われる数字をそれぞれ 2, 5, 6 にしたところだ。だからと言ってこの時点で、256 の数ができたわけではないことを注意しておこう。

6: 行目の代入で、f(0), f(1), f(2) の順に百の位、十の位、一の位の数が決まるので、ここではじめて 256 という数ができたことになる。すると 7: 行目の代入は、各位を逆順にした数になっていることに気付くだろう。従って、ここでは 652 という数ができたのだ。それが 9:, 10: 行目で表示される。

マクロはこれだけだが、このネタをもう少し調理しよう。

2 進数というのはコンピュータではよく使われる、1001011 や 11111011 といった数のことだ。われわれは普段 10 進数を使っているので、これらの数が 77 や 251 だということに気付かないものだ。コンピュータは 1 バイトという単位を使うが、1 バイトは 8 ビットである。え？　何を言ってるの分からないって？　つまりこういうことだ。1 ビットとは 0 か 1 だけが使える桁のことである。だから 8 ビットとは 0 か 1 が使える桁が 8 つあることを指している。そしてこれが 1 バイトになる。要するに 1 バイトで 00000000 〜 11111111 までの数が扱えるのだ。マクロ例は 1 バイトの 2 進数がいくつになるかを調べるものだ。

programming list [BinToDec.vba]

```
 1 : Sub BinToDec()
 2 : Dim f(8), dec As Integer
 3 :
 4 :     f(0) = 1: f(1) = 1: f(2) = 1: f(3) = 1
 5 :     f(4) = 1: f(5) = 0: f(6) = 1: f(7) = 1
 6 :
 7 :     dec = 2 * (2 * (2 * (2 * (2 * (2 * (2 * f(0) + f(1)) + _
 8 :         f(2)) + f(3)) + f(4)) + f(5)) + f(6)) + f(7)
 9 :
10 :     Sheet1.Cells(2, 1) = dec
11 : End Sub
```

プログラムは一部が美しくないけれど、正確な値を出してくれる。美しくないのは 4:, 5: 行目で、ご丁寧にもひとつひとつの配列変数に代入している。この代入によって 2 進数の 11111011 がセットされたことになる。実はこのセットの仕方が美しくない原因になっているのだが気にしないでおこう。

さて、10 進数なら十の位が $10^1$、百の位が $10^2$、千の位が $10^3$、...である。2 進数では十の位とは呼ばないので、下の位から順に $2^0$ の位、$2^1$ の位、$2^2$ の位、...である。従って 11111011 を 10 進数にするには $2^7\mathrm{f}(0)+2^6\mathrm{f}(1)+\cdots+2^0\mathrm{f}(7)$ の計算—ほらね、美しくないでしょ。だって指数と配列番号が一致してないもの—をすればよい。しかしコンピュータはべき乗の計算を得意としていないのだ。だから $2^7\mathrm{f}(0)+\cdots$ の計算をさせたければ、`dec = 2*2*2*2*2*2*2*f(0)+ ···` と書かなくてはならない。

そこで 7: – 8: 行目のような、中途半端に因数分解した式を使うことになるのだけれど、これがコンピュータにとって実に好都合なのだ。7: 行目の末尾に " _" が付いているのは、**VBA** 特有の記号である。ここの式は 1 行に納めるには少々長過ぎたので 2 行に分けたのだ。記号 " _" を使えば、長い式を複数行に分けて書くことができるので重宝する。

では、`dec = 2*(2*(2*(2*(2*(2*f(0)+ ··· )+f(7)` と `dec = 2*2*2*2*2*2*2*f(0)+ ··· +f(7)` を較べてほしい。掛け算をする回数がかなり減っていることが分かるはずだ。実際、前者は 7 回の掛け算で済んでいるが後者は 28 回も掛け算をしている。コンピュータは掛け算に時間がかかってしまうものである。この程度の計算では恩恵が目立たないが、7: – 8: 行目に使われた式のほうが、掛け算の回数が少ないから計算時間が短縮されるのだ。

ところで話題をマクロ `[BinToDec.vba]` に戻すことにしよう。ここでは 2 進数の 11111011 を 10 進数に直しているが、実行すると 251 が返ってくるはずだ。だが、コンピュータのプログラムでは整数型の最高位が 1 なら、その数は負の数であるとすることがある。その場合は整数型の 11111011 は $-5$ を表すのだ。8 桁では分かりにくいだろうから、4 桁の 2 進数を例にとろう。

最高位が 0 か 1 によって正の数と負の数の区別がつくので、0111 は正の数で 1111 は負の数である。しかし 1111 は、正の数 0111 にマイナスの意味で最高位の 1 を付けているわけではない。1111 は $-1$ である。なぜなら $1111+1=0000$ だから、$1111=-1$ であることが分かるというものだ。2 進数は 1111, 1110, 1101, 1100, 1011, ...という具合に減るから、これらは 10 進数で $-1$, $-2$, $-3$, $-4$, $-5$, ...を表している。

もし 4 桁の 2 進数が用意されれば、全部で $2^4=16$ 通りの数が表せる。その内、0xxx で表される正の数が 8 通り、1xxx で表される負の数が 8 通りだ。0xxx の正の数には 0 も含まれるので、正の数は 0 から 7 までだが、1xxx の負の数は $-1$ から $-8$ までとなる。お分かりかな？　よって 4 桁の 2 進数が扱える数は $-8$ から 7 までである。

ちょっと前に、2 バイト—これは 16 桁の 2 進数に相当している—なら $256^2 (= 2^{16}) = 65536$ 通りの数が使えるから、正負で折半して $-32768 \sim 32767$ の整数が扱えると言ったことを覚えているかい。正の数がひとつ少ない気がするのは、やはり負の数が $-1$ から始まるのに対し、正の数は 0 から始まるからだ。