8...の散歩道

8.1　新たな符号？

方程式 $x^2 = -1$ は一般に解くことができない。なぜなら、どんな数 x も 2 乗すると $x^2 \geq 0$ となり、負の値になることがないからである。しかし、数学には新しい定義をする技がある。そして、定義があとから矛盾を生じないものなら、それは数学の世界で大きな顔をすることができる。$x^2 = -1$ となる数を定義してしまおう。いまのところは仮想的な数（imaginary number）の意味で、その数を i と書くことにする。i は 2 乗すると -1 になるので

$$i^2 = -1$$

という数が定義できたことになる。

これまで数の計算においては、数値だけでなく符号にも注意を払ってきたことだろう。$(+)\times(-) = (-)$ であるとか $(-)\times(-) = (+)$ であるとか。いまの定義よりここに $(i)\times(i) = (-)$ が加わることになった。ただ、こういう書き方をすると i が符号のように見えるが、実際、符号の性質も併せ持つ。$(-)\times(-) = (+)$ が $(-1)\times(-1) = (+1)$ の計算から符号だけ取り出したと見れば、$(i)\times(i) = (-)$ は $(i1)\times(i1) = -1$ から符号だけ取り出したと見えなくもない。もっとも、$i1$ は文字式の約束から $1i$ と書いた上で 1 を省略して i とするので、数の $i1$ と符号の i の区別がつかないようなものである。

このことから、符号の立場で積を考えると

$$(+)\times(+)\,, \qquad (+)\times(-)\,, \qquad (+)\times(i)\,,$$
$$(-)\times(-)\,, \qquad (-)\times(i)\,, \qquad (i)\times(i)\,.$$

の組み合わせが考えられる。その結果、現れる符号は順に

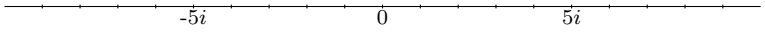
$$(+)\,,\quad (-)\,,\quad (+i)\,,\quad (+)\,,\quad (-i)\,,\quad (-)\,.$$

である。$+i$ は i じゃないよ。あくまでも符号の立場で見ているので、それらが混ざることはないのだ。さて、$(+)$ が 3 個、$(-)$ が 3 個、(i) が 2 個、合計 8 個の符号が生き残ったことになる。

2

強引ではあるけれど、8 の小道を散歩するにふさわしい数値だ。i がひとつ足りない気がするかもしれないね。大丈夫、そう感じる人のためにこの先の道に i が現れるものがある。それもちょっと強引かもしれないけど。

さて、符号 i がついた数は**虚数**と呼ばれる立派な数だ。虚数は虚数専用の数直線（虚数軸）に表すことができる。



計算も可能だ。$i^2 = -1$ に注意すれば、文字式並みの計算ができる。

$$\begin{aligned} 2i \times 3 &= 6i \\ 2i \times 3i &= 6i^2 = -6 \end{aligned}$$

Python3 では虚数が扱える。ただし **Python3** の虚数符号は i ではなく、`j`（または `J`）を使うことになっている。数学で単位に j を使うときは、**四元数（しげんすう）** $a + bi + cj + dk$ に使うときである。四元数とは

$$i^2 = j^2 = k^2 = -1, \quad ij = -ji = k, \quad jk = -kj = i, \quad ki = -ik = j$$

という性質をもつ単位 i, j, k で構成される数である。たしかに j^2 も -1 なので辻褄は合うけれど、ちょっと数学感覚と違う点に注意しよう。

[py script]

```
>>> 1j*1j
(-1+0j)
>>> 1j**2
(-1+0j)
```

Python3 では `j` はあくまでも符号の扱いである。そのため `j*j` では符号どうしの演算、というより定義されてない変数の演算となってしまい、エラーが返ってくる。そこで $i \times i$ なら `1j*1j` と書くのだが、結果は `(-1+0j)` と表示される。`1j**2` でも同じだ。虚数符号 `j` の係数は `0` だから、実際は -1 のことである。

実数符号 $+$、$-$ と虚数符号 i は混じることがないから、実数符号つきの数 a と虚数符号付きの数 bi の和は $a + bi$ と書く以外にない。つまり、和の形でひとつの数を表すので、( ) で囲まれていると思ってよい。このような数は**複素数**と呼ばれる。この中には $a = 0$ や $b = 0$ である数も含まれるから、bi だけでも a だけでも複素数と呼んでよい。ただ、`bj` なら複素数であることが見て分

かるけれど、a だと複素数であることが見てわからない。数学の定義では a だけでも複素数と見るが、**Python3** では複素数は実数であっても複素数として扱いたい都合がある。そこで、一旦複素数で計算が始まると、j は省略できないのである。

[py script]
```
>>> 2*3, 2*3j, 2j*3j
(6, 6j, (-6+0j))
```

複数の計算をまとめて行った。実数どうしの計算では 0j はつかず、虚数の計算は j がついて虚数であることが分かり、たとえ結果が実数になっても虚数計算では 0j と ( ) がつく。これらの結果がさらに ( ) で囲まれているのは、**Python3** 特有の表示をしているからだ。