第8週の庭いじり

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

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

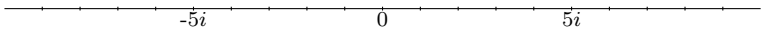
の組み合わせが考えられる。その結果、現れる符号は順に

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

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。

強引ではあるけれど、第 8 週の庭いじりにはふさわしい数値だ。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。

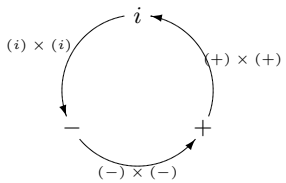
さて、符号 i がついた数は**虚数**と呼ばれる立派な数だ。虚数は虚数専用の数直線（虚数軸）に表すことができる。



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。

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

感覚的に $2i \times 3 = 6i$ というのは、i を負の符号 $-$ に見立てた計算 $-2 \times 3 = -6$ と違いはない。また $2i \times 3i = -6$ というのも、同じく $-2 \times -3 = +6$ と違いはない。$(i) \times (i) = (-)$ であることが $(-) \times (-) = (+)$ であることに対応しているようなものである。余談ながら、この続きが $(+) \times (+) = i$ であったなら、



のような閉じた関係になって面白かったかもしれない。しかし $(+) \times (+) = (+)$ である。ちなみに数学では**四元数（しげんすう）**$a + bi + cj + dk$ なる数が考えられている。それは

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

という性質をもつ単位 i, j, k で構成される数である。こちらは i, j, k が巡回して気持ちよい。

さて、話を戻そう。虚数単位 i は実数と馴染むように見えるが、$2i + 3$ は $2i + 3$ でしかなく決して $5i$ にはならない。したがって a, b を実数とすると、一方に虚数単位が付いた—たとえば b に虚数単位を付けて bi とした—場合、$a + bi$ はこのままでひとつの値と見るしかない。$\sqrt{2} + 3$ がこのままでひとつの値であるように。

このような数は**複素数**と呼ばれる。かりに $b = 0$ のときは $a + bi = a$、すなわち実数を表すので、実数は複素数に含まれると考えてよい。このように i を含む複素数は、a と b が相容れないため、

$a+bi$ と書く代わりに $(a,\ b)$ と書くことができる。a は実数そのものなので複素数の**実部**、b は虚数単位 i に付属する実数なので複素数の**虚部**と呼ぶ。

複素数を $(a,\ b)$ で表したとき、複素数は **Haskell** にとってタプルを連想させる。実際、**Haskell** は

(ghci env.)
```
Prelude> (3, 2)
(3,2)
Prelude> fst (3, 2)
3
Prelude> snd (3, 2)
2
```

のようにタプルを扱うので、タプルは数学の計算に使えそうだ。しかし残念なことに、複素数どうしのつもりで 2 数を足そうとしても

(ghci env.)
```
Prelude> (3, 2)+(5, 1)

<interactive>:6:1: error:
    • Non type-variable argument in the constraint: Num (a, b)
      (Use FlexibleContexts to permit this)
    • When checking the inferred type
        it :: forall a b. (Num a, Num b, Num (a, b)) => (a, b)
```

のように跳ね返されてしまう。無理をすれば

(ghci env.)
```
Prelude> (fst (3, 2) + fst (5, 1), snd (3, 2) + snd (5, 1))
(8,3)
```

のように計算できないこともないけれどね。

実は、この無理を道理に変える力が **Haskell** にはあるのだ。つまり、型を自分で定義できる。たとえば、上で行ったような計算ができる “複素数型” は次のように定義する。

[hs script]
```
data Complex a = Complex a a deriving (Show)

cplus :: (Num a) => Complex a -> Complex a -> Complex a
(Complex x y) `cplus` (Complex z w) = Complex (x+z) (y+w)
```

そうすると、`mod` で剰余の計算ができたように、`cplus` で複素数の計算ができるのだ。

(ghci env.)
```
*Main> Complex 3 2 `cplus` Complex 5 1
Complex 8 3
```

ほらね。同じ計算結果だ。でも、この型はお世辞にも見やすいとは言えない。`Complex 8 3` で

は、引数 `8 3` をとる関数 `Complex` と思われるかもしれない。関数は大文字で始まらない、という **Haskell** のルールがあるから、その心配はないのだが、もっと見やすくならないものだろうか。

しかし、これ以上の説明は抜きだ。だって、複素数の計算はあらかじめ **Haskell** に組み込まれているんだから、わざわざ自分で定義するまでもないのだ。けれど型を定義できるというのは、特別な計算規則が必要になったとき、強い見方になってくれる。さっきの四元数って **Haskell** で定義されているんだろうか？　定義されていなければ、自分で作ればよい。