マンデルブロ集合

再帰についてもう少し続けよう。漸化式は再帰の関係を表すと言った。たとえば

$$z_n = z_{n-1}^2 + c \quad (c \text{ は定数}), \qquad z_0 = 2, \quad c = -1$$

のような漸化式は再帰関数にできる。

```
func recursion(n: Int) -> Int {
    if n > 0 {
        return recursion(n: n-1) * recursion(n: n-1) - 1
    } else {
        return 2
    }
}
```

みたいに。この場合は `recursion(n: 5)` などとして、5 番目の値を求められる。ところでこの漸化式は、値が際限なく大きくなることがすぐに推察できる。馴染みある関数 $y = x^2 - 1$ の関係と大差ないからである。

しかし、この漸化式を実数ではなく**複素数**で考えると、見える景色は大きく異なる。たとえば

$$z_n = z_{n-1}^2 + c \quad (c \text{ は定数}), \qquad z_0 = 3 - i, \quad c = 1 + 2i \qquad (※)$$

としてみよう。複素数は中学校で扱わないものだが、実数と**虚数単位** i を組み合わせた数で、たとえば $-2 + i$、$\sqrt{2} - \dfrac{4}{3}i$、$5 + 2\pi i$ などは複素数である。要するに、中学校までに登場した数 a, b と虚数単位 i を組み合わせて $a + bi$ の形にした数である。i は文字式の文字みたいなものだ。ただし i には一つだけ特殊事情がある。それは、$\boxed{i^2 = -1}$ になることである。

だから（※）において、z_1 の計算は

$$z_1 \quad \to \quad (3-i)^2 + (1+2i) \quad = \quad (9 - 6i + i^2) + (1 + 2i) \quad = \quad 9 - 4i$$

となって、$z_2 \quad \to \quad (9-4i)^2 + (1+2i) \quad = \quad \ldots$ と z_n の計算は延々続く。まあ、こんな具合だ。複素数の世界は奥が深いけれど、今回の話はこれだけの知識でいけるので、先へ進むことにしよう。

（※）の例は、実際に計算を続ければわかるように、計算結果として現れる $a+bi$ の形の数は、$|a|$, $|b|$ ともにどんどん大きくなっていく。このような状態を z_n は**発散する**ということにする。一方、

$$z_n = z_{n-1}^2 + c \quad (c \text{ は定数}), \qquad z_0 = 0.5, \quad c = 0.5i$$

の場合は[1]、

[1] $z_0 = 0.5 + 0i$、$c = 0 + 0.5i$ でいずれも複素数。実数 a は $a + 0i$ のことで、（実数）$\subset$（複素数）である。

n	0	1	2	3	4
z_n	0.5	$0.25+0.5i$	$(-0.188)+0.75i$	$(-0.527)+(0.219)i$	$(0.230)+(0.269)i$

のようになって、発散するように見えない。実際、発散しないであるところに凝集していくようになる。このような状態を z_n は**収束する**ということにする。

つまり複素数の範囲で考えると、z_n はときに発散し、ときに収束することがわかる。したがって、ある漸化式に与える z_0, c によって複素数の組 $(z_0,\ c)$ は、発散する集合と収束する集合に分かれることになる。このような集合を**マンデルブロ集合**[2]という。

では、発散する集合と収束する集合はどのように棲（す）み分けられるのだろうか。もう少し条件を絞って考えよう。そこで、初期値である z_0 は 0（すなわち $0+0i$）とし、z_0 に対して定数 c を変えたとき、どんな c の値で z_n の発散/収束が認められるかを調べることにする。プログラムは、複素数 c による発散/収束の分布を表示するものである。

ところで、このプログラムは **Create New Project...** で表示されるダイアログにおいて、“iOS – App playground” ではなく、“iOS – App” を選択して作成しよう。それは複素数を扱うため、`ComplexModule` の `import` を必要とするのと[3]、処理時間が長いので playground では具合が悪いのである。

[mandelbrot.~~playground~~]

```
 1 import SwiftUI
 2 import ComplexModule
 3
 4 struct ContentView: View {
 5     var body: some View {
 6         @State var arrdata = dotsdata()
 7
 8         GeometryReader { geometry in
 9             ForEach(0 ..< arrdata.count, id: \.self) { index in
10                 let col = switch (arrdata[index].1) {
11                     case 1...20: Color.blue
12                     default:     Color.black
13                 }
14                 Circle()
15                     .fill(col)
16                     .frame(width: 1, height: 1)
17                     .offset(x: arrdata[index].0.x, y: arrdata[index].0.y)
18             }
```

[2]ブノワ・マンデルブロ (1924–2010)：フランスの数学者、経済学者。

[3]`import ComplexModule` を記述した場合、`ComplexModule` がダウンロードされてなければ “Search package collections” と促されるので、指示に従って導入する（または Xcode のメニューから “File” > “Add Package Dependencies...” をたどり、“swift-numerics” を追加。その際、“Add to Target” には App を選択）。

```
19          }
20      }
21
22      func dotsdata() -> [(CGPoint, Int)] {
23          var arrdata: [(CGPoint, Int)] = []
24          for im in stride(from: -1.25, to: 1.25, by: 0.01) {
25              for re in stride(from: -2, to: 0.5, by: 0.01) {
26                  arrdata += [(CGPoint(x: 100*re + 200, y: 100*im + 125),
                                     recurnum(c: Complex<Double>(re, im)))]
27              }
28          }
29          return arrdata
30      }
31
32      func recurnum(c: Complex<Double>) -> Int {
33          var z = Complex<Double>(0, 0) // z_0 := 0+0i
34          for i in 1 ..< 21 {
35              z = z * z + c
36              if (z.length > 2) {
37                  return i // 発散（-> blue）
38              }
39          }
40          return -1 // 収束（-> black）
41      }
42  }
```

マンデルブロ集合については、書籍や Internet でさまざまな情報が得られるので、ここでプログラムについて詳細は述べず、簡単に捕捉的な説明に留めておこう。

基本的には、Project 作成時に用意されたコードを書き換えるのだが、今回は `Path {` ではなく `GeometryReader {` を用いた。描画を `path` で線を結ぶのではなく点で描くためだが、座標を正しく扱いたいことも理由である。その際、`for` 文ではなく `ForEach` 文を用いたのは、どうやら `GeometryReader` では `for` 文が使用できないようだ。

`ForEach` 文は `for` 文より使い勝手はよいけれど、一定のパターンを処理する仕様のようなので、定数 c の発散/収束を都度計算して、その度に点を描画する方法はうまくないようである（“〜ようである” ばっかでごめんね）。そこで、定数 c の発散/収束の様子はあらかじめ配列に収めることにして、配列データを `ForEach` により `GeometryReader` で描画している。

`func dotsdata()` が定数 c の発散/収束の様子を配列に収める関数だ。見てわかると思うが、定数 c は**複素平面**上の、左上 $(-2,\ 1.25) = -2 + 1.25i$ から右下 $(0.5,\ -1.25) = 0.5 - 1.25i$ まで[4]の矩形すべてにわたって 0.01 間隔で調べている。つまり、計算する c の総数は 250×250 である。データは、複素平面座標を View の座標に変換して格納している。

[4]数学では一般に、複素数 $-2 + 1.25i$ は複素平面では座標 $(-2,\ 1.25)$ と表す（y 座標の i はつけない）。

それぞれの c における発散/収束の判定をするのが `func recurnum()` だ。ここでは z_n の初期値は $z_0 = 0$ である。具体的に判定の様子を説明しておこう。

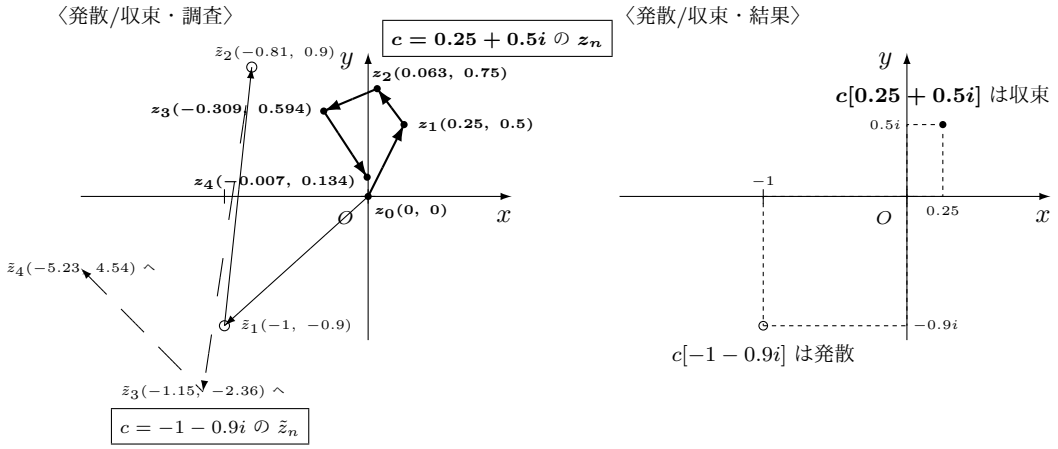
〈発散/収束・調査〉の図は、ある一つの c に対する z_n の変化を複素平面に表示したもので、

$$z_n = z_{n-1}^2 + (0.25 + 0.5i),\quad z_0 = 0 \qquad \text{と} \qquad \tilde{z}_n = \tilde{z}_{n-1}^2 + (-1 - 0.9i),\quad \tilde{z}_0 = 0$$

の 2 例を示した（ともに $_{n=0}$ から $_{n=4}$ まで）。

〈発散/収束・結果〉の図は、それぞれの c が複素平面上のどの位置にあるかを、発散を ○、収束を ● でプロットしたもので、先の 2 例について示した。View 画面に表示されるときは、それぞれ青点と黒点で描画される。

この二つは異なる複素平面であることに注意されたい。〈発散/収束・調査〉の図は、`recurnum()` が行う計算をイメージしたもので描画とは関係ない。プログラムによって実際に描画されるのが〈発散/収束・結果〉の図、ということである。



$c = 0.25 + 0.5i$、すなわち $z_n = z_{n-1}^2 + (0.25 + 0.5i),\quad z_0 = 0 + 0i$ から始めた場合、z_n は

$$z_0[0] \quad \to \quad z_1[0.25 + 0.5i] \quad \to \quad z_2[0.063 + 0.75i] \quad \to \quad z_3[-0.309 + 0.594i] \quad \to \quad \cdots$$

と変化する。これは収束するパターンだ（左図の y 軸まわり）。また、$c = -1 - 0.9i$、すなわち $\tilde{z}_n = \tilde{z}_{n-1}^2 + (-1 - 0.9i),\quad \tilde{z}_0 = 0 + 0i$ から始めた場合、$\tilde{z}_n$ は

$$\tilde{z}_0[0] \quad \to \quad \tilde{z}_1[-1 - 0.9i] \quad \to \quad \tilde{z}_2[-0.81 + 0.9i] \quad \to \quad \tilde{z}_3[-1.15 - 2.36i] \quad \to \quad \cdots$$

と変化する。これは発散するパターンである（左図の第 2 象限から第 3 象限）。

プログラムで発散/収束を判定するには、一定の基準が必要だ。再帰計算は永久に続けられるが、20 回ほど繰り返せば十分である。その間に z_n が原点 O から 2 以上の距離になったら、もう z_n は原点から離れる一方である[5]。その際、収束するときは `-1` を格納するが、発散するときは計算回数である `i` を格納するようにした。現状 `GeometryReader` での描画は黒点か青点の区別しかしないので、発散するときは `0` を格納すれば十分である。

しかし発散回数を記録しておくと、発散回数による点の色分けが可能になる。もし Internet などでマンデルブロ集合を見たことがあれば、カラフルな模様を見たことだろう。それらは、発散の回数による色分けの産物である。このプログラムも `switch` 文の `case` を細かく分けたり、たとえば `arrdata[index].1 % 8` に変更するなどして色分けができる。好みの色分けで実行するとよいだろう。

また、描画範囲を狭く—発散/収束の境目付近に—して、計算間隔を `0.001` など細かくすれば、境目付近の様子を描画できる。いろいろ試すことを勧めたい。

z_n 自体は再帰的だが、プログラムは再帰関数を用いていない。発散がわかれば計算を打ち切るためだ。しかし、再帰関数でなくとも計算量は多いので、結果の表示に少々時間がかかる[6]。気長に待ってもらいたい。また、描画は画面左上を基準に表示されるが、中央に表示させるには `arrdata` に格納する座標変換の式を少しいじる必要がある。

[5]原点 $O(0,\ 0)$ と $z_n(re,\ im)$ との距離は三平方の定理により $\sqrt{re^2 + im^2}$ 。

[6]Xcode を立ち上げて最初に実行する場合は、“Build » Simulator 起動 » ... ” など数段階の手続きのため時間を要す。また、機器のメモリ量にも左右される。表示まで数分程度のこともある。