2.6 確率

確率の考えは日常において、降水確率やらコンビニの商品陳列やらに用いられている。数学は何でもそうだが、確率の元となる考えはものすごく単純で、

$$(\text{事象 A が起こる確率}) = \frac{(\text{事象 A の場合の数})}{(\text{全事象の場合の数})}$$

である。**事象**は物事のことであるから、全体の物事に A が占める割合が確率の定義である。定義が簡単だからといって確率の計算が簡単である保証はない。実は逆にものすごく難しい。難しくしている原因は、全事象に含まれる物事ひとつひとつが起こる可能性が**同様に確からしい**ことを前提にするからである。

サイコロを転がしたとき、1 の目から 6 の目までの出方は同様に確からしいだろうか。たとえばサイコロがいびつな形のせいで、2 の目は多く出るが 5 の目はあまり出ない、というのではいけない。均質な材質で、重心が中心にあるのが理想だ。本物のサイコロは 1 の目が大きく掘られているから、1 の面は少し軽くないか？

しかし、理想的なサイコロであっても、つまりどの目が出ることも同様に確からしいとしても、6 の目が 4 回続けて出ることはあるのだ。

Swift が生成する乱数が均質な数の列を発生させるとしても、そうそう完璧な分布ができるわけではない。ただ、**試行**が十分多く行われるならば、サイコロはどの目も $\frac{1}{6}$ の割合で出現するのである。シミュレーションをしてみよう。

[Probability.playground]

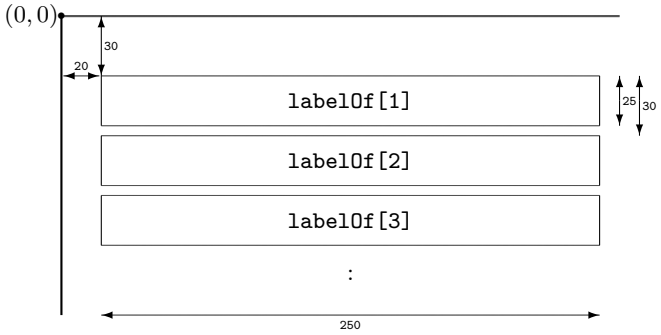
```
import UIKit
import PlaygroundSupport // *

class ViewController: UIViewController {

    var countOf = [0, 0, 0, 0, 0, 0, 0]
    var ratioOf = [Float](repeatElement(0.0, count: 7))
    var labelOf = [UILabel()]
    var counter = 0
    var diceButton = UIButton()
    var trial = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()

        labelOf = []
        for i in 0...6 {
            labelOf += [UILabel(frame: //__
```

```
                                CGRect(x: 20, y: 30*i, width: 200, height: 25))]
        }
        for i in 1...6 {
            labelOf[i].text = "\(i): \(countOf[i]) 回　（\(ratioOf[i])）"
            view.addSubview(labelOf[i])
        }

        trial = UILabel(frame: //__
                     CGRect(x: 20, y: 250, width: 120, height: 25))
        trial.text = "合計 0 回"
        view.addSubview(trial)

        diceButton = UIButton(frame: //__
                           CGRect(x: 150, y: 250, width: 50, height: 25))
        diceButton.addTarget(self, action: //__
                            #selector(dispResult), for: .touchUpInside)
        diceButton.setTitle("dice", for: UIControl.State.normal)
        diceButton.setTitleColor(UIColor.black, for: UIControl.State())
        view.addSubview(diceButton)
    }

    @objc func dispResult() {
        counter += 1
        let n = Int(arc4random()) % 6 + 1
        diceButton.setTitle(String(n), for: UIControl.State.normal)
        countOf[n] += 1

        for i in 1...6 {
            ratioOf[i] = Float(countOf[i])/Float(counter)
            labelOf[i].text = "\(i): \(countOf[i]) 回（\(ratioOf[i])）"
        }
        trial.text = "合計 \(counter) 回"
    }

    // (timer)

}

let viewController = ViewController()                  // *
viewController.view.backgroundColor = UIColor.white    // *
PlaygroundPage.current.liveView = viewController       // *
PlaygroundPage.current.needsIndefiniteExecution = true // *
```

近年のプログラミングはイベントに対してアクションを起こすことが基本のようだ。今回は diceButton を押す—イベントが発生する—たびに、サイコロの目を示し出た回数を記録する—アクションを起こす—仕様のプログラムである（サイコロは 1 個しか振らないのだから dieButton だろう、というツッコミはやめてね）。同時に、出た目の割合も計算している。

各サイコロの目について集計をするため、変数は配列で用意するのがよいだろう。回数を記録する変数 `countOf` は整数で、割合を記録する変数 `ratioOf` は実数で初期化している。6 行目と 7 行目で初期化の仕方が異なっているのは整数と実数のせいではなく、単に配列の初期化の方法が何通りかあるので例を示しただけだ。`countOf` と同じように、`ratioOf = [0.0, 0.0, ...]` でもまったく問題ない。配列名が不自然に見えると思うが、あとで `countOf[1]` のような使い方をするので、この名前なら "1 の目の回数" がわかりやすいと思っただけである。

ところで配列の要素が 7 個あるのに気づいただろうか。プログラム上は 6 個あればよいのだが、サイコロの目の数と配列番号を一致させたほうがわかりやすいだろうと思い、ハナから `countOf[0]` は使わないつもりだった。使わないものを用意するのはメモリの無駄だが、これっぽっちの無駄ならかまわないだろう。



プログラムは最初に、目が出た回数を表示するためにラベルを 6 個配置している。以前、似たようなことをしているので何をしているかわかるはずだ。ただ、今回はラベルは等間隔に配置できるので `for` 文で繰り返すことができた。でも `labelOf[0]` は使わないので、属性を持たせるラベルは `labelOf[1]` から `labelOf[6]` だけである。その下にもうひとつ、試行回数を表示するラベルも設置している。コードは長ったらしいが「定義して → 属性を与えて → 配置する」のはいつもどおりである。

`diceButton` には `UIButton()` を代入して属性を与えているが、たとえばボタンにタイトルを表示させる仕方などが、ラベルに文字を表示させる仕方と異なっている。Swift はなぜ、そんな仕様にしてるんだろう？ ま、いいか。

`diceButton` に枠すら表示しないのは理由がある。確率の実験をする場合は、とてつもなくたくさんの試行をしたいのだが、ボタンを押し続けるのは大変だろう。すぐあとでプログラムを修正して、ボタンを押すことがないようにする予定だからだ。もっともボタンに枠を表示させて悪いことはないので、見た目を気にするなら 32 行目の下に属性を追加しよう。たとえば、太さが 1 の黒線

を引くなら

```
diceButton.layer.borderWidth = 1
diceButton.layer.borderColor = UIColor.black.cgColor
```

を属性として与えよう。他にも細かな設定がいろいろできるので、必要なことは各自で調べてほしい。

ところで、ラベルやボタンはプログラムを書いて配置しているが、実は Xcode には、部品をマウスでちょいちょいと配置できる仕組みがある。Storyboard という。本当はちょいちょいというほど楽ではないけれど、視覚的に行えるのは利点だろう。でも私は、ラベルをたくさん配置するならプログラムを書くほうが楽だと思うけどね。好みの問題なので、興味があれば Storyboard についても調べてみるとよいだろう。

おっと、プログラムに戻ろう。ボタンを押したときのアクションは関数 `dispResult` が受け持つ。乱数を発生させて乱数の値に応じて `countOf[n]` に `1` を加算するだけである。同時に割合も計算することにしている。

ボタンを押して正常に動作することが確認できたら、さっそく `// (timer)` の位置に以下のコードを追加挿入してみよう。ボタンは Swift に押してもらうのだ。でも実際に押すのではなく、ボタンを押す代わりに自動でループを回すんだけどね。

```
var timer: Timer!
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(true)
    timer = Timer.scheduledTimer(timeInterval: 1, //__
                  target: self, selector: #selector(update), //__
                  userInfo: nil, repeats: true)
    timer.fire()
}

override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(true)
    timer.invalidate()
}

@objc func update(tm: Timer) {
    dispResult()
}
```

`timeInterval: 1,` となっているところは、1 秒間隔でタイマーが作動することを意味している。ここを `1/10` にすれば 0.1 秒、`0.05` にすれば 0.05 秒間隔となるが、CPU 動作や画面表示には相応の時間がかかるものなので劇的に速くサイコロが振れるわけではない。それでも、確率 $\frac{1}{6}$ であることが何を意味するかは、シミュレーションを見ていれば想像がつくというものだ。

計算上求めることができる確率は言わば数学的確率で、実験観察から得られる確率—これは起こった事象が試行に占める割合—とは多少異なる。しかし、試行回数を “十分大きく” すると双方の確率がほぼ等しいことが示される。これは**大数の法則**と呼ばれる。

では、試行回数が “十分大きい” とはどれぐらいの回数を指すのだろう。1,000 回？、100 万回？、10 億回？（←どれも unicorn だね）。実はとくに定めがあるわけではない。実験観察から得られた値が、計算上の値にほぼ等しくなるまで繰り返した回数が十分大きいということなのである。ずるいと思うかい。

まあ適度に大きい試行回数、たとえば 6,000 回サイコロを振ればどの目も 1,000 回程度は出るだろう。でも実際は、すべてがきっちり 1,000 回になることはない。1 の目は 1,058 回かもしれないし、6 の目は 971 回かもしれない。ただ、6,000 回振ることを 1 セットとし、数千・数万セット繰り返せば 1 の目がちょうど 1,000 回出ることが、1,058 回出るとか 983 回出るとかより多いはずである。じゃあ、ちょうど 1,000 回出ることを中心にして、(999 回, $1,001$ 回)、(998 回, $1,002$ 回)、(997 回, $1,003$ 回)、$\ldots$ がペアで同じ割合で少なくなるかというとそうでもない。

1,000 回を中心に左右対称の分布になれば**正規分布**なのだが、この場合は違う。なぜなら、6,000 回振って 1 の目が少ない極端な例は 1 回も出ないことだが、多い極端な例は 6,000 回出ることだからである。つまり、0 回から 6,000 回までの分布において 1,000 回がいちばん多く出現する分布は左右対称ではない。そのような分布は**二項分布**と呼ばれる。`@objc func update(tm: Timer)` を次のものに差し替えて試してみよう。

```
var countOne = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
@objc func update(tm: Timer) {
    if counter % 30 == 0 {
        switch countOf[1] {
            case 0...9: countOne[i] += 1
            default: countOne[10] += 1
        }
        print(countOne)
        countOf = [0, 0, 0, 0, 0, 0, 0]
    }
    dispResult()
}
```

これはサイコロを 30 回振ることを 1 セットとし、1 の目が各セットで何回出たかを `var countOne` に記録するものである。1 の目は 0 回から 30 回まで出る可能性があるが、10 回以上はひとくくりにしてカウントしてある。`[Probability.playground]` をちょっと修正しただけのプログラムなので、Live View にはサイコロを振る様子がシミュレートされたままである。合計が 30 の倍数に

なるごとにカウント数がリセットされるのが観察できるはずだ。で、リセットのせいで出た目の割合は役立たずになってしまったけど。

でも、いま知りたいのはそこではなく、30 回 1 セットの中で 1 の目がどれだけ出たかであるから、変数 `countOne` の記録を見なくてはいけない。それは Debug Area に逐次表示される。ちょっと分かりづらい表示なのだが、30 回のセットが 3 回行われると、たとえば

```
[1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0]
```

のような結果が示される。これは各セット 30 回の試行中に、1 の目が 0 回出た（要するに 1 回も出ない）ときが 1 度、1 の目が 3 回出たときが 1 度、1 の目が 5 回出たときが 1 度、という意味である。Live View の合計値が 30 の倍数になるごとに積み上げられた記録が表示されるだろう。ただし条件が `if counter % 30 == 0` のままでは、試行回数が 0 の時点でも `if` 文が真となるので、いきなり `[1, 0, 0, ...]` が Debug Area に現れる。`if` 文に条件を追加するか、あらかじめ `var countOne = [-1, 0, 0, ...]` とすればよいだろう。

ところで、サイコロは 30 回振れば 1 から 6 の目は 5 回ずつ出ることが期待されるので、理論的にはどの目も 1 セットで 5 回出ることが多いはずだ。でも、4 回だけ出たり 6 回だけ出たりすることも、5 回出ることと同じぐらい起こりそうだ。正確な計算の仕方は、たぶん高 1 数学で学ぶだろう。

ex. 1 下の表は Microsoft Excel で計算した理論値である。シミュレーションの結果と比べてみよう。その際は、`ratioOf[i]` の計算と表示を修正しよう。

回数	0 回	1 回	2 回	3 回	4 回	5 回	6 回	7 回	8 回	9 回	10 回以上
割合	0.004	0.025	0.073	0.137	0.185	0.192	0.160	0.110	0.063	0.031	0.020